Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Coordenação de Pós-Graduação em Informática

UNIFIC: Uma API para a Unificação da Programação entre Diferentes GUIs

Lílian Gibson Silva

Campina Grande - PB
Dezembro - 1993

Lílian Gibson Silva

UNIFIC: Uma API para a Unificação da Programação entre Diferentes GUIs

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

Área de Concentração: Ciência da Computação

Jacques Phillipe Sauvé

(*Orientador*)

Campina Grande

Dezembro - 1993

S586u Silva, Lilian Gibson.
UNIFIC : uma API para a unificação da programação entre diferentes GUIs / Lilian Gibson Silva. - Campina Grande, 1993.
195 f.

Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1993.
"Orientação : Prof. Dr. Jacques Philippe Sauvé".
Referências.

1. Interfaces Gráficas do Usuário (GUIs). 2. Interface de Programação de Aplicações (API). 3. UNIFIC - Unificação da Programação. 4. Dissertação - Informática. I. Sauvé, Jacques Phillipe. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 004.5(043)

UNIFIC - UMA API PARA A UNIFICAÇÃO DA PROGRAMAÇÃO ENTRE DIFERENTES GUIs

LÍLIAN GIBSON SILVA

DISSERTAÇÃO APROVADA EM 21.12.1993

JACQUES PHILIPPE SAUVÉ, Ph. D.
Orientador

ULRICH SCHIEL, Dr.
Componente da Banca

MARIA DE FÁTIMA Q. V. TURNELL, Ph. D.
Componente da Banca

Campina Grande, 21 de dezembro de 1993

A meus pais e irmãos.

Agradecimentos

Ao professor Jacques Philippe Sauvé, pela orientação e apoio que muito contribuíram para a realização deste trabalho.

A Capes, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, que financiou parte deste trabalho.

A todos os professores que, direta ou indiretamente, contribuíram para a conclusão deste trabalho.

Aos funcionários do Departamento de Pós-graduação de Sistemas e Computação da UFPB, em especial a Ana Lúcia Guimarães e a Zeneide Medeiros de Britto Lyra, pela eficiência e prestatividade com as quais pude contar durante o desenvolvimento deste trabalho.

A minha família e amigos, pelo apoio e paciência no transcorrer deste trabalho.

Resumo

Atualmente, um problema enfrentado pelos programadores de aplicações com Interfaces Gráficas do Usuário (GUIs) é a falta de portabilidade desse tipo de aplicação para diferentes ambientes gráficos. Este trabalho apresenta a especificação, implementação e tutorial de uma Interface de Programação de Aplicações (API), denominada UNIFIC, que unifica a programação entre diferentes GUIs. As aplicações desenvolvidas com a API UNIFIC apresentam portabilidade inerente no seu código fonte para diferentes ambientes gráficos.

Abstract

Nowadays, a problem that must be solved by designers of Grafical User Inferface (GUI) applications is their portability to different graphical environments. This work presents the specification, implementation and tutorial for an Application Programming Interface (API), named UNIFIC, that makes the development of GUI applications uniform across platforms. UNIFIC's applications are portable to different graphical environments without having to rewrite code.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 7 - Conclusão

Bibliografia

Apêndice A - Funções da Biblioteca UNIFIC

Apêndice B - Comandos do Arquivo de Recursos

Capítulo 1

1. Introdução

Nos últimos anos, o interesse na pesquisa e desenvolvimento de interfaces para aplicações, voltadas para as necessidades do usuário, vem aumentando. O principal motivo desse interesse é a constatação de que, a qualidade da interface do usuário de uma aplicação pode ser tão importante, para o seu sucesso, quanto a sua funcionalidade. No entanto, o desenvolvimento de interfaces agradáveis e produtivas, do ponto de vista do usuário, não é uma tarefa fácil para o projetista de aplicações.

1.1. Motivação

Numa tentativa de garantir o nível de qualidade das interfaces do usuário, surgiu nos anos 80 um novo conceito de interface: a Interface Gráfica do Usuário.

1.1.1. Interfaces Gráficas do Usuário

A Interface Gráfica do Usuário, ou *Graphical User Interface* (**GUI**), é um tipo de interface rica em informações visuais, cujos símbolos gráficos de alta definição são usados para representar objetos em vídeos mapeados a *bits*[1]. Cada objeto da GUI possui aparência e comportamento (*look-and-feel*) uniformes, de acordo com o grupo ao qual pertence. Janelas, menus e ícones são exemplos de alguns destes objetos (Figura 1.1). Este tipo de interface facilita

[1] Vídeos mapeados a *bits*, ou vídeos *bit-mapped*, são vídeos que utilizam milhares de pequenos pontos (*pixels*) para compor caracteres, números e gráficos.

a comunicação homem-máquina pois a execução de aplicações, seleção de comandos ou realização de outras tarefas, é sempre feita através da seleção e ativação dos objetos gráficos, cujas imagens são facilmente associadas à sua funcionalidade.

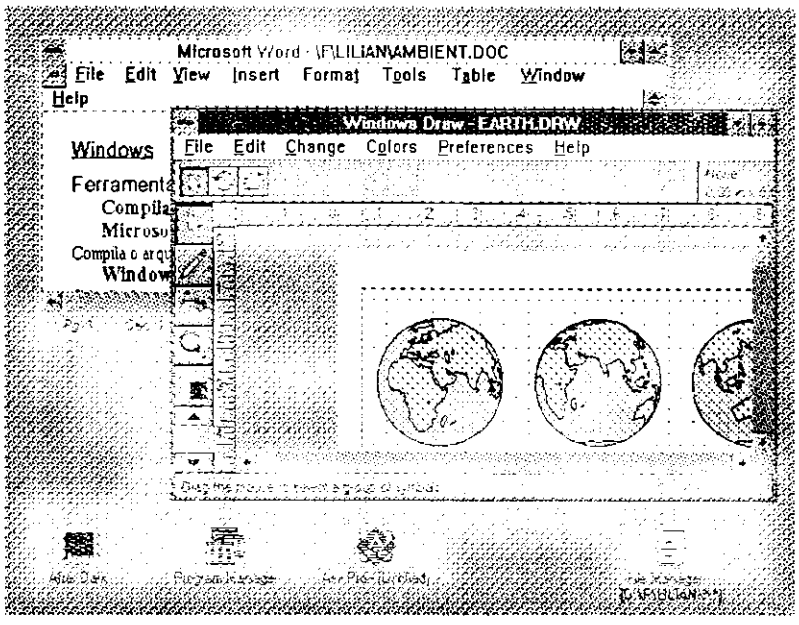

Figura 1.1 Interface Gráfica do Usuário

1.1.1.1. Vantagens

As principais vantagens das GUIs quando comparadas com as interfaces baseadas em caracteres[2], do ponto de vista do usuário, são: facilidade de uso, interação visual instantânea e uniformidade.

• **Facilidade de uso**: os símbolos gráficos que representam os objetos das GUIs são rapidamente reconhecidos e memorizados pelo usuário, tornando mais fácil a identificação da funcionalidade associada a eles.

• **Interação Visual Instantânea**: é oferecida uma interação física direta, normalmente através de um *mouse*, entre o usuário e os objetos da GUI. Esta interação gera uma

---

[2] As interfaces baseadas em caracteres utilizam apenas caracteres alfabéticos, numéricos e de pontuação para apresentar ao usuário informações na tela. No máximo, podem fazer uso de caracteres semigráficos.

retroalimentação visual instantânea que reduz o tempo de aprendizagem e fornece ao usuário uma "sensação" de controle sobre o computador.

• **Uniformidade**: a GUI oferece um mecanismo padrão de controle para cada objeto. Assim operações semelhantes são sempre executadas da mesma forma. A modificação do tamanho de uma janela, por exemplo, é feita da mesma forma em qualquer aplicação. Isto permite que um usuário que esteja familizarizado com o *look-and-feel* de uma determinada GUI possa utilizar, sem problemas, qualquer aplicação desenvolvida para esta GUI.

1.1.1.2. Arquitetura

Do ponto de vista do programador, uma GUI oferece toda infra-estrutura para o desenvolvimento de aplicações com interfaces gráficas. Geralmente, quatro componentes destacam-se em uma GUI (Figura 1.2): guia de estilo, API, sistema de janelas e modelo de imagem.

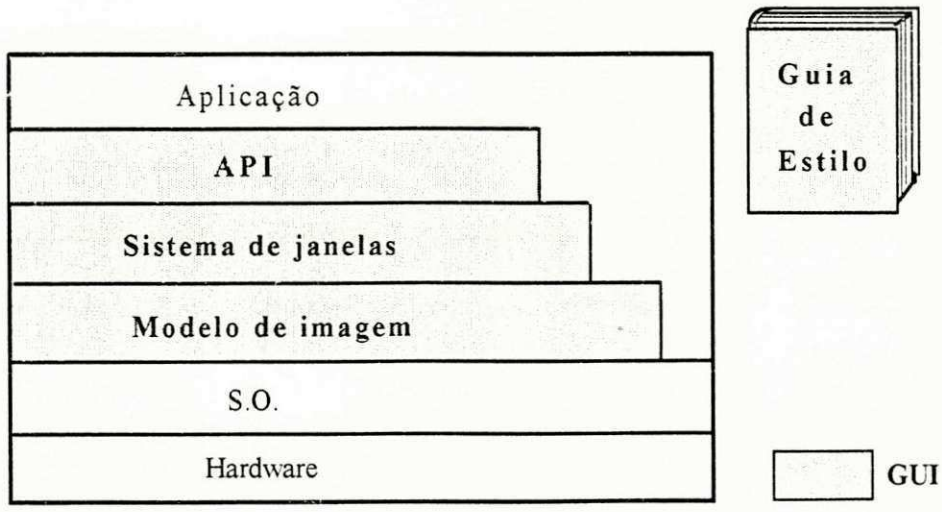

Figura 1.2 Arquitetura em camadas de uma GUI

• **Guia de estilos**: é um documento que especifica como deve ser a aparência e o comportamento dos objetos da GUI. O guia de estilos de uma GUI deve ser seguido pelo

programador para garantir a homogeneidade do *look-and-feel* entre as aplicações desenvolvidas.

• **Application Programming Interface** (API): a API de uma GUI é composta por bibliotecas de funções pré-definidas, estruturas de dados e comandos, colocados à disposição do programador para permitir a criação e a gerência dos objetos da GUI (ex.: janelas ou menus). Geralmente a documentação das bibliotecas da API mostra apenas como são as interfaces das funções, mas não especifica como as funções devem ser combinadas para que se obtenha uma padronização do *look-and-feel* nas aplicações desenvolvidas. Portanto, o programador deve usar as bibliotecas da GUI em conformidade com o guia de estilos.

• **Sistema de Janelas**: é o sistema que torna possível a visão simultânea de várias aplicações em execução, em diferentes janelas de uma mesma tela. Esse sistema é responsável pela criação e gerência das janelas e pelo controle da disposição dos objetos da GUI no vídeo. Além disso, gerencia a entrada/saída de dados entre os dispositivos e as aplicações, bem como a alocação dos recursos requeridos por cada aplicação.

• **Modelo de Imagem**: especifica como a GUI acessa o vídeo, definindo como os caracteres e os gráficos serão apresentados na tela.

1.1.1.3. Portabilidade entre Interfaces Gráficas do Usuário

Atualmente, a maioria das plataformas de *hardware* existentes no mercado já possui pelo menos uma GUI disponível. A tabela 1.1 apresenta algumas dessas GUIs.

Observando-se a tabela 1.1, pode-se constatar que não existe uma GUI padrão independente de máquina, que seja amplamente aceita pelo mercado. O motivo é que a maioria dos fabricantes buscou uma solução própria. A falta de padronização das GUIs resulta, a princípio, em dois problemas: falta de portabilidade e a perda da cultura de programação.

GUI	HARDWARE	SISTEMA OPERACIONAL
Macintosh	Apple	MacOS
Windows	IBM PCs e compatíveis	MS-DOS, NT
Presentation Manager	IBM PCs e compatíveis	OS/2, UNIX
Motif	(vários)	UNIX, MS-DOS e OS/2
NextStep	(vários)	UNIX, MS-DOS e OS/2
Metaphor	(vários)	UNIX, MS-DOS e OS/2
NewWave	HP	UNIX
Windows Dec	DEC	UNIX
OPEN LOOK	Sun	UNIX

Tabela 1.1 Exemplo de GUIs disponíveis comercialmente e suas plataformas.

• **Falta de Portabilidade**: as aplicações escritas para GUIs proprietárias, como as GUIs Windows e Macintosh, geralmente não são portáveis para outras arquiteturas ou sistemas operacionais. Existe uma dependência de *hardware* ou de *software* próprias das aplicações escritas para estas GUIs, que torna crítico o transporte das aplicações. Além disso, o transporte das aplicações escritas para GUIs não proprietárias também pode não ser tão simples, principalmente, se as camadas inferiores da arquitetura das GUIs diferirem consideravelmente.

• **Perda da Cultura de Programação**: a filosofia e as ferramentas utilizadas para o desenvolvimento das aplicações variam tanto de uma GUI a outra que, raramente, a cultura de programação obtida em uma GUI é útil a outra. Conseqüentemente, o projetista tem que aprender a programar as particularidades de cada uma delas.

Esses dois problemas dificultam o desenvolvimento de aplicações que precisam ser transportadas para diferentes GUIs e podem causar um grande impacto no custo final de desenvolvimento dessas aplicações.

Segundo uma estimativa da Microsoft, para se desenvolver uma aplicação com GUI é necessário 50% a mais do tempo que seria utilizado, para se desenvolver a mesma aplicação com uma interface baseada em caracteres [Gray91]. Cerca de um terço desse tempo é gasto apenas com particularidades da API e do *look-and-feel* da GUI escolhida. Já para transportar uma aplicação de um ambiente gráfico para outro - por exemplo, transportar uma aplicação Windows para ambiente Motif - utiliza-se mais um terço do tempo necessário para se desenvolver a aplicação em um único ambiente gráfico. Portanto, o tempo gasto para se desenvolver uma mesma aplicação para dois ambientes gráficos, normalmente, é o dobro do tempo que seria necessário, se a aplicação fosse desenvolvida com uma biblioteca de interface baseada em caracteres. Isto significa que o desenvolvimento de aplicações com GUIs geralmente tem um custo elevado que aumenta à medida em que a aplicação precisa ser transportada para um novo ambiente gráfico.

Esses são alguns dos fatores que motivam os esforços para que se estabeleça uma GUI padrão independente de máquina. Como a idéia de uma GUI padrão está longe de ser alcançada devido à concorrência mercadológica e ao grande número de bases instaladas com diversas GUIs, ferramentas que possibilitam a execução de uma aplicação em vários ambientes gráficos já estão sendo objetos de estudo. Essas ferramentas são extremamente importantes, visto que reduzem o tempo e, conseqüentemente, os custos de programação necessários ao transporte das aplicações entre diferentes GUIs.

1.2. Objetivos do Trabalho

Este trabalho propõe o desenvolvimento de uma biblioteca de funções, denominada **biblioteca UNIFIC**, que unifique a programação entre diferentes GUIs. O seu principal objetivo

é permitir que as aplicações desenvolvidas com esta biblioteca sejam portáveis entre diferentes ambientes gráficos, de tal forma que o código fonte da aplicação, não precise sofrer nenhuma modificação para ser executado na plataforma destino. Por exemplo, uma aplicação desenvolvida com a biblioteca UNIFIC em ambiente Windows, poderá ser executada em um ambiente Motif apenas com a recompilação de seu código fonte no ambiente destino.

Os objetivos específicos deste trabalho são:

• Prover uma API baseada em um modelo de programação que unifique a programação entre diferentes GUIs.

• Mostrar a viabilidade da API UNIFIC através da sua implementação para um ambiente gráfico.

1.3. Importância do Trabalho

A importância desse trabalho deve-se ao fato de qualquer aplicação que venha a ser desenvolvida com a biblioteca UNIFIC, terá portabilidade inerente no seu código fonte para diferentes ambientes gráficos. A biblioteca UNIFIC tornará as diferenças de programação entre as várias GUIs, transparentes para o programador, pois proverá uma forma unificada de programação para GUIs distintas. Isso eliminará o esforço, antes necessário, para transportar aplicações entre diferentes GUIs, reduzindo, conseqüentemente, os custos de desenvolvimento das aplicações.

1.4. Trabalhos Relacionados

Já existem, atualmente, alguns produtos disponíveis comercialmente que permitem o desenvolvimento de aplicações portáveis para diferentes ambientes gráficos. A biblioteca XVT é

um exemplo desse tipo de produto. Essa biblioteca, desenvolvida pela XVT Software Inc., oferece uniformidade de programação para os ambientes Windows, Macintosh, OS/2 Presentation Manager, OSF/Motif e OPEN LOOK, além de incluir suporte para os ambientes UNIX, DOS e VMS baseados em caracteres. As aplicações desenvolvidas com a biblioteca XVT podem ser transportadas para quaisquer dos ambientes citados acima, sem que o seu código fonte necessite ser re-escrito.

Durante a pesquisa bibliográfica constatou-se que, apesar da disponibilidade comercial de alguns produtos para a unificação da programação entre diferentes GUIs, do ponto de vista acadêmico existem carências de informações a respeito desta área de estudo. Isso deve-se ao fato dos fabricantes desse tipo de produto não divulgarem informações sobre o processo de unificação e nem o código fonte das suas APIs. Este trabalho contribui para o preenchimento dessa lacuna, podendo vir a ser utilizado como referência básica no desenvolvimento de estudos futuros na área de unificação da programação entre diferentes GUIs.

1.5. Organização do Trabalho

Este capítulo apresenta uma visão geral das GUIs, sua arquitetura e alguns problemas de portabilidade que motivaram o desenvolvimento deste trabalho, bem como seus objetivos e importância.

O capítulo 2, *Caracterização da Biblioteca UNIFIC*, apresenta uma visão geral da biblioteca UNIFIC e define seus requisitos básicos e específicos.

O capítulo 3, *Ambientes Gráficos Unificados*, seleciona as GUIs que participarão inicialmente do processo de unificação, apresenta suas arquiteturas e caracteriza os modelos de programação adotados pelas respectivas GUIs.

O capítulo 4, *Modelos de Programação da UNIFIC*, define o modelo de programação unificado da biblioteca UNIFIC.

O capítulo 5, *Programando com a Biblioteca UNIFIC*, apresenta a estrutura básica de uma aplicação UNIFIC e explica, através de exemplos, como o programador deve utilizar a biblioteca UNIFIC.

O capítulo 6, *Implementação da Biblioteca UNIFIC*, descreve os detalhes da implementação da biblioteca UNIFIC em um dos ambientes gráficos selecionados.

No capítulo 7, *Conclusão*, são avaliados os objetivos do trabalho, apresentadas as conclusões finais e sugestões para trabalhos futuros.

O Apêndice A descreve a interface e a funcionalidade das funções colocadas à disposição do programador através da API UNIFIC.

O Apêndice B apresenta o formato geral dos comandos que podem ser utilizados pelo programador no arquivo de recursos das aplicações UNIFIC.

Capítulo 2

2. Caracterização da Biblioteca UNIFIC

Este capítulo discute as principais características da biblioteca UNIFIC. Inicialmente será apresentada uma visão global da biblioteca. Em seguida, serão definidos seus requisitos básicos, tais como: transparência, portabilidade, abrangência, facilidade de programação e *look-and-feel* da plataforma destino. Finalmente serão apresentados os requisitos específicos que englobam aspectos de interação da UNIFIC com as GUIs unificadas e com os usuários.

2.1. Visão Global

A biblioteca UNIFIC é uma camada de *software* que oferece um modelo de programação unificado para o desenvolvimento de aplicações com GUIs (Figura 2.1). O modelo unificado de programação tem como principal objetivo permitir que o programador desenvolva aplicações portáveis para vários ambientes gráficos, utilizando o mesmo conjunto padrão de funções.

A fim de oferecer ao programador um conjunto padrão de funções consistente e genérico, é necessário que a biblioteca UNIFIC atenda a alguns requisitos. Esses requisitos podem ser classificados em dois grupos: requisitos básicos e requisitos específicos. Os **requisitos básicos** são os que devem ser atendidos pela UNIFIC do ponto de vista das aplicações (Figura 2.1a). Já os **requisitos específicos** são os que devem ser atendidos pela biblioteca UNIFIC do ponto de vista das GUIs unificadas (Figura 2.1b). Este é o assunto a ser tratado nas duas próximas seções.

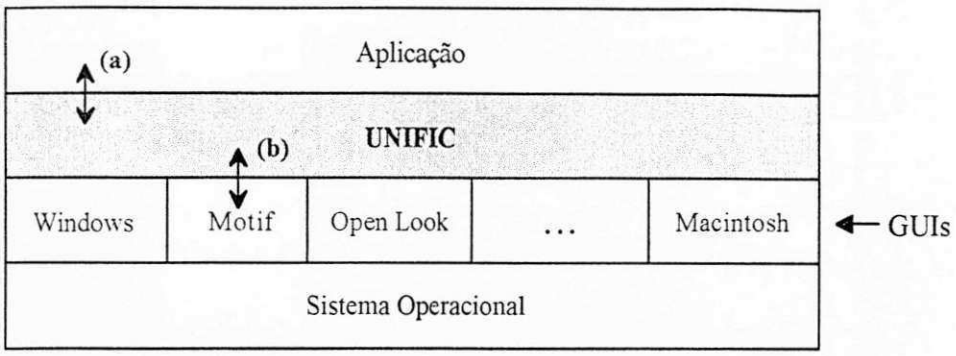

Figura 2.1 Visão global da biblioteca UNIFIC.

2.2. Requisitos Básicos

Os requisitos básicos atendidos pela biblioteca UNIFIC são: transparência, portabilidade, abrangência, facilidade de programação e *look-and-feel* da plataforma destino.

2.2.1. Transparência

A biblioteca UNIFIC deve tornar transparente, para o programador, as diferenças existentes entre os modelos de programação das GUIs unificadas. O conjunto padrão de funções da biblioteca UNIFIC deve permitir que o programador desenvolva aplicações, utilizando sempre a mesma filosofia, independentemente de o ambiente gráfico de desenvolvimento ser, por exemplo, Windows, Motif, Open Look ou Macintosh.

2.2.2. Portabilidade do Código Fonte

Uma aplicação desenvolvida com a biblioteca UNIFIC deve ser portável para vários ambientes gráficos, de tal forma que não seja necessário efetuar nenhuma modificação no seu código fonte. O programador deve apenas recompilar a aplicação no ambiente destino.

2.2.3. Abrangência

O modelo de programação definido para a biblioteca UNIFIC deve ser o mais genérico possível, de tal forma que possa ser implementado sem modificações nos mais diversos ambientes gráficos. Isso garantirá que novas GUIs possam ser facilmente acrescentadas ao conjunto de GUIs unificadas pela biblioteca UNIFIC.

2.2.4. Facilidade de Programação

A API UNIFIC deve permitir que, durante a fase de desenvolvimento das aplicações, o programador possa utilizar as práticas de projeto e codificação disponíveis atualmente. O modelo de programação adotado pela UNIFIC deve ser simples e de fácil utilização.

2.2.5. Look and Feel da Plataforma Destino

A biblioteca UNIFIC não deve impor um *look-and-feel* particular a suas aplicações. Ao invés disto, as aplicações UNIFIC devem herdar o *look-and-feel* da plataforma destino. Isto é, se uma aplicação UNIFIC for executada, por exemplo, em um ambiente Motif, ela herdará o *look-and-feel* da GUI Motif. No entanto, se a mesma aplicação for executada em um ambiente Windows, ela deverá assumir automaticamente o *look-and-feel* da GUI Windows.

2.3. Requisitos Específicos

Os requisitos específicos abrangem, fundamentalmente, três aspectos: o primeiro engloba as necessidades de interação entre a UNIFIC e o sistema operacional; o segundo considera as necessidades de comunicação entre aplicações desenvolvidas com a UNIFIC; finalmente, o terceiro trata das formas de interação entre a UNIFIC e o usuário.

2.3.1. Interação entre a UNIFIC e o Sistema Operacional

Toda GUI interage diretamente com o sistema operacional quando necessita acessar os recursos do computador. As operações de alocação de recursos, manipulação de arquivos, acesso à memória e aos *drivers* de dispositivos são sempre realizadas através do sistema operacional utilizado pela GUI. A biblioteca UNIFIC deve, portanto, permitir que suas aplicações realizem este tipo de operação em qualquer ambiente gráfico.

A fim de atender aos requisitos de interação com o sistema operacional, a UNIFIC deve considerar aspectos de comunicação com o gerente de processos, o gerente de memória e o gerente de arquivos.

2.3.1.1. Comunicação com o Gerente de Processos

A UNIFIC deve permitir que a aplicação se comunique com o gerente de processos utilizado pela GUI. É a partir da interação da aplicação com o gerente de processos que é possível, a uma aplicação, por exemplo, mudar a prioridade de execução de certas tarefas ou criar processos para executar outras aplicações.

2.3.1.2. Comunicação com o Gerente de Memória

A UNIFIC deve permitir que suas aplicações aloquem memória dinamicamente. Para tornar isto possível, as aplicações UNIFIC necessitam comunicar-se com o gerente de memória utilizado por cada GUI. Apesar do tipo de gerente de memória utilizado pelas GUIs influenciar no conjunto de funções necessárias para se gerenciar memória em um ambiente gráfico, é possível identificar um conjunto básico de funções que deve estar disponível na UNIFIC. Este conjunto deve permitir que o programador realize, no mínimo, operações de alocação, realocação e liberação de áreas de memória.

2.3.1.3. Comunicação com o Gerente de Arquivos

A UNIFIC deve permitir que as aplicações realizem operações sobre arquivos. As funções que geralmente estão à disposição do programador, na maioria das GUIs, para este propósito são funções específicas para criar, abrir e fechar arquivos, além de funções para leitura, escrita de dados e posicionamento do apontador dentro do arquivo acessado. Normalmente, além das funções citadas acima, funções para a realização de operações sobre diretórios também devem ser colocadas à disposição do programador. Essas funções devem permitir ao programador, pelo menos, obter e mudar o diretório de trabalho das aplicações.

2.3.2. Comunicação entre Aplicações UNIFIC

O sistema de janelas de uma GUI permite que várias aplicações, executadas concorrentemente ou em paralelo, compartilhem ao mesmo tempo a tela de um computador. O fato de as aplicações dividirem o mesmo espaço fisico para mostrar o resultado de suas execuções, faz surgir, naturalmente, o desejo de trocar dados entre estas aplicações. As GUIs permitem que as aplicações se comuniquem entre si, a fim de que possam ser realizadas operações de troca de dados entre aplicações. Portanto, a biblioteca UNIFIC deve oferecer mecanismos que tornem possível para o programador realizar o compartilhamento padronizado de dados entre suas aplicações.

2.3.2.1. Mecanismos de Troca de Dados

Na maioria das GUIs, os mecanismos de troca de dados entre aplicações que se encontram disponíveis são o *clipboard* e a troca de mensagens.

Clipboard

O *clipboard* é uma área de acesso comum a todas as aplicações, onde os dados a serem compartilhados podem ser colocados temporariamente. A fim de realizar a troca de dados através deste mecanismo, o usuário deve selecionar os dados que deseja compartilhar na aplicação origem (Figura 2.2a). Em seguida deve ativar o comando que transfere os dados selecionados da aplicação para o *clipboard* (o comando *copy* de um editor de texto, por exemplo) (Figura 2.2b). Finalmente o usuário deve se dirigir à aplicação destino e ativar o comando que copia os dados do *clipboard* para a aplicação (o comando *paste* de um editor de texto, por exemplo) (Figura 2.2c). Este mecanismo de troca de dados sempre requer a intervenção do usuário para que se efetue a transferência dos dados. Para que a UNIFIC dê suporte à transferência de dados através do *clipboard*, funções específicas para remover, copiar ou ler dados do *clipboard*, por exemplo, devem ser colocadas à disposição do programador.

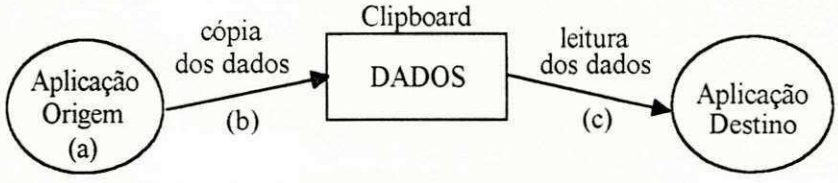

Figura 2.2 Troca de dados através do clipboard.

Troca de Mensagens

A troca de dados entre aplicações também pode ser feita através de um mecanismo de troca de mensagens. Esse tipo de troca de dados baseia-se no modelo cliente-servidor. Nesse esquema, é necessário que se estabeleça uma conversação entre a aplicação que requer os dados e a aplicação que pode fornecê-los. Inicialmente a aplicação cliente deve enviar uma mensagem de pedido para a aplicação servidora, a fim de especificar os dados desejados (Figura 2.3a). Se a aplicação servidora possuir a informação desejada, uma mensagem de resposta com os dados

requeridos é enviada de volta para a aplicação cliente (Figura 2.3b). Para que a UNIFIC dê suporte a este mecanismo, ela deve oferecer ao programador um conjunto de funções que permitam as aplicações trocarem mensagens entre si.

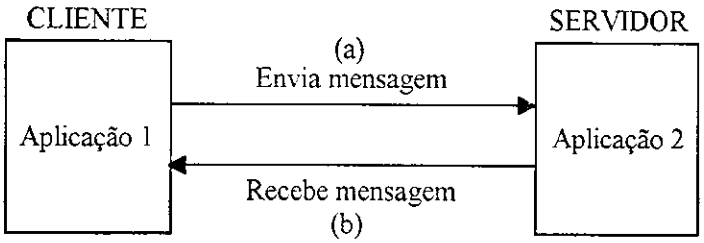

Figura 2.3 Troca de dados através de troca de mensagens.

2.3.3. Interação entre a UNIFIC e o Usuário

A interação entre a biblioteca UNIFIC e o usuário pode ocorrer de três formas: através da entrada de dados orientada a eventos, através da saída gráfica e através dos objetos da GUI.

2.3.3.1. Entrada de Dados Orientada a Eventos

Quando um usuário deseja interagir com uma GUI, ele normalmente o faz digitando informações através do teclado ou selecionando e ativando os objetos da GUI por meio de um apontador de dispositivo, como por exemplo, um *mouse*. Cada ação realizada pelo usuário é detectada pela GUI e transformada em um evento. Um evento é um conjunto de informações que notifica a aplicação de algum fato do seu interesse ou indica a necessidade da realização de alguma ação particular à aplicação. Os eventos recebidos por uma aplicação podem ter sido gerados através da atividade de algum dispositivo, como a interação do usuário com a GUI, por exemplo, pelo próprio sistema de janelas ou até mesmo por outras aplicações com GUIs. Cada evento é gerado em relação a um objeto da GUI e deve ser enviado para a aplicação a qual o objeto pertence.

Todas as GUIs modernas utilizam um sistema de entrada de dados orientado a eventos. Isto significa que o fluxo de controle da aplicação não é determinado pela própria aplicação, mas sim pelos eventos gerados pelo usuário. Os eventos podem ocorrer a qualquer hora, em qualquer ordem e em qualquer objeto, já que o usuário pode mover o apontador do *mouse*, entrar com dados pelo teclado, mover e redimensionar janelas, ou ativar funções através dos objetos da GUI. Esse tipo de entrada de dados oferece alta prioridade ao usuário, permitindo que ele interfira em qualquer parte do processo.

Em um esquema de entrada de dados orientado a eventos, normalmente o gerente de eventos da GUI é o responsável pela interpretação e envio dos eventos para a aplicação apropriada. A figura 2.4 apresenta uma visão geral dos passos que normalmente ocorrem neste processo.

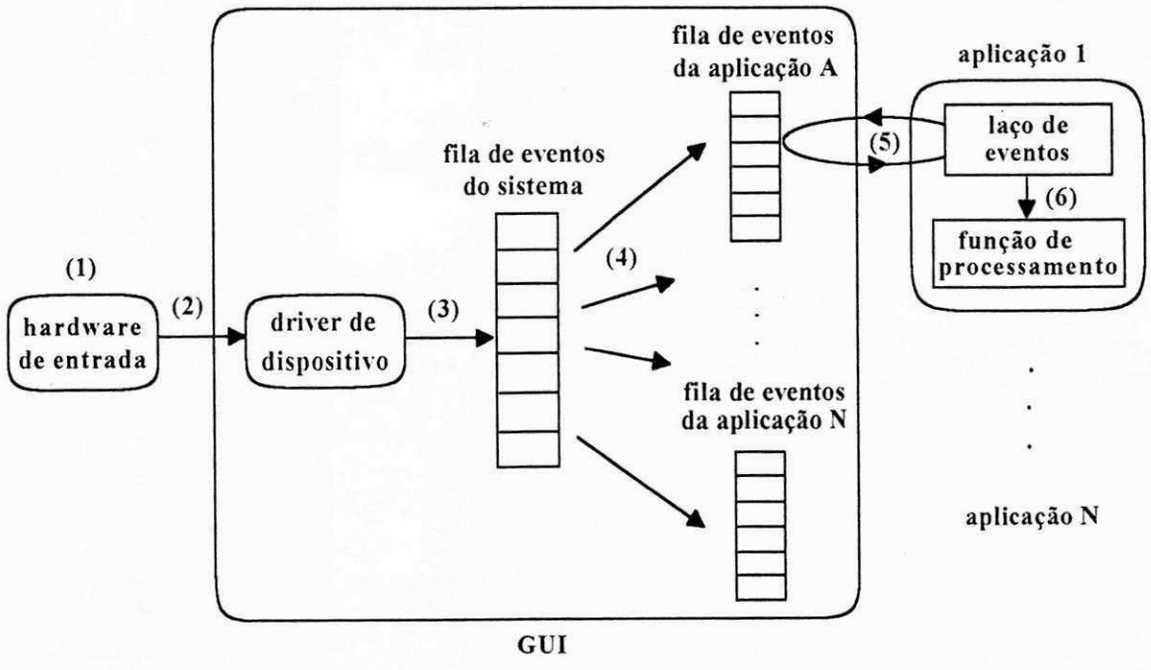

Figura 2.4 Gerente de eventos.

(1) O usuário interage com a GUI por meio do apontador ou teclado.

(2) Os *drivers* de dispositivos detectam a ação do usuário e armazenam as informações

relacionadas à ação em uma estrutura de dados denominada registro de eventos[3].

(3) O evento é então enviado para a fila de eventos do sistema.

(4) O sistema de gerência de eventos da GUI retira o evento da fila de eventos do sistema e o envia para a fila de eventos da aplicação[4] apropriada.

(5) Quando a aplicação está pronta para receber um evento, ela o lê da sua fila de eventos, normalmente através de um laço específico.

(6) A aplicação então, identifica o evento recebido e o envia para processamento.

A biblioteca UNIFIC deve oferecer meios para que a aplicação interaja com o gerente de eventos das GUIs unificadas, a fim de obter seus eventos da fila de eventos da aplicação (passo 5). Também, deve ser possível para a aplicação, identificar o objeto para o qual o evento foi enviado e encaminhá-lo para a função de processamento adequada.

2.3.3.2. Saída Gráfica

Toda saída gerada por uma aplicação com GUI é gráfica. Isto se deve ao fato dos vídeos mapeados a *bits*, utilizarem o *pixel* como a sua menor unidade de acesso. Esse tipo de acesso permite que caracteres de texto e imagens sejam gerados da mesma forma. Isto é, graficamente. A saída gráfica oferece várias vantagens, tanto para o programador de aplicações, quanto para o usuário. Do ponto de vista do programador passam a existir fontes[5] de vários tipos e tamanhos, além de uma perfeita compatibilidade entre imagens gráficas e texto. Do ponto de vista do usuário, as informações gráficas tornam as aplicações mais atrativas visualmente e apresentam informações complexas de uma forma mais simples.

---

[3] O registro de eventos geralmente contém informações sobre a hora de ocorrência do evento, a posição do *mouse*, o estado do teclado, o código da tecla ou botão do *mouse* pressionado, e o dispositivo a partir do qual o evento foi gerado.

[4] Cada instância da aplicação possui a sua própria fila de eventos.

[5] Um fonte é um modelo de caracteres com uma determinada aparência, estilo, tamanho e espessura, utilizado para mapear os códigos de caracteres em letras, na tela do computador.

O grande problema a ser resolvido pela maioria das GUIs para permitir a realização das operações de saída gráfica é definir a metodologia de especificação dos parâmetros necessários a este tipo de operação. Normalmente as operações de saída gráfica exigem do programador a definição de um grande número de atributos. Se todos estes atributos tivessem que ser definidos de uma forma particular em cada operação gráfica, a tarefa de programação para gerar saída gráfica poderia ser inviável. Para diminuir o número de parâmetros exigidos pelas funções gráficas e permitir que um determinado conjunto de atributos, já definido, possa ser reutilizado, várias GUIs exigem que os atributos do gráfico sejam especificados antecipadamente, através de funções especiais. Quando uma função gráfica é então chamada, ela utiliza os atributos definidos previamente pelo programador. O dispositivo destino, a largura e o estilo da linha usados pelo gráfico, as cores e os fontes são exemplos de alguns dos atributos de um gráfico.

A biblioteca UNIFIC deve oferecer meios para que o programador defina, antecipadamente, os atributos desejados para as operações gráficas e também deve colocar à disposição do programador um conjunto de funções que realize operações de saída gráfica de forma compatível em qualquer ambiente destino.

2.3.3.3. Objetos da Interface do Usuário

Como visto no capítulo 1, as GUIs fazem uso de objetos da GUI para oferecer uma interface amigável para o usuário de aplicações. A maioria dos aspectos relacionados à aparência e ao comportamento dos objetos da GUI é pré-definido pelas próprias GUIs, para garantir a compatibilidade do *look-and-feel* entre suas aplicações. Portanto, a UNIFIC deve interagir com as GUIs unificadas para permitir a criação e gerência de seus objetos, de acordo com o *look-and-feel* da GUI destino.

A seguir será feita uma descrição dos objetos que são normalmente encontrados na maioria das GUIs e que podem ser colocados à disposição do programador na biblioteca UNIFIC. São eles: janelas, ícones, barras de rolamento da janela, apontador, cursor, menus, controles e quadros de diálogo.

Janelas

As janelas são áreas retangulares que dividem a tela do computador em várias regiões lógicas independentes (Figura 2.5). Dessa forma, cada janela pode executar tarefas distintas ao mesmo tempo, sem que as atividades de uma tenham influência nas outras.

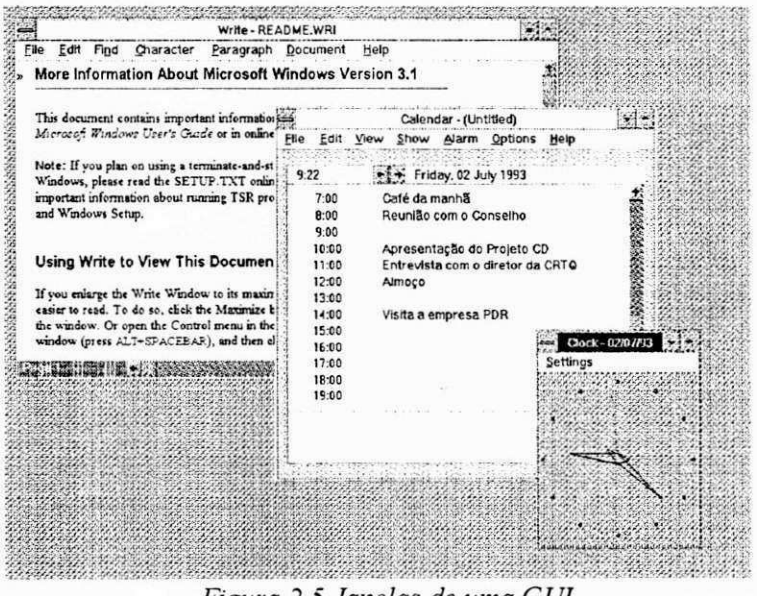

Figura 2.5 Janelas de uma GUI.

Este tipo de objeto oferece um grande potencial, tanto para o usuário, quanto para o programador. Considere, por exemplo, que uma janela seja associada a uma aplicação. Nesse caso o usuário pode acompanhar os resultados de várias aplicações em execução simultaneamente, nas janelas de uma mesma tela. Já do ponto de vista do programador, as janelas têm a finalidade de organizar os demais objetos da interface do usuário e oferecer uma área de comunicação com o usuário. A janela é o principal objeto da interface do usuário.

Ícones

Os ícones são símbolos gráficos, usados para representar uma aplicação, um comando ou algum dado (Figura 2.6). Os ícones usados para representar aplicações têm a finalidade de distinguir uma aplicação das outras. Esse tipo de ícone está sempre associado à janela principal[6] de uma aplicação e pode ser visualizado sempre que esta janela for fechada. Já os ícones de comandos ou dados, representam opções que o usuário necessita acessar com frequência. Exemplos de ícones usados com esta finalidade são os que permitem a maximização do tamanho de uma janela que podem ser encontrados na borda de uma janela.

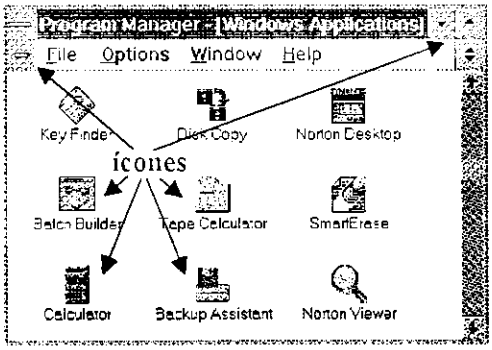

Figura 2.6. Ícones de aplicações e ícones de comandos

Barras de Rolamento da Janela

A barra de rolamento de uma janela é um objeto de forma retangular, composto por dois ícones de comandos e um indicador de posição (Figura 2.7).

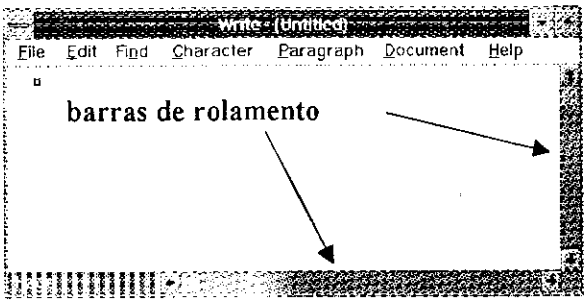

Figura 2.7 Janela com barras de rolamento.

---

[6] A janela principal de uma aplicação é a primeira janela visível ao usuário quando a aplicação é executada. Esta janela é a diretriz básica que vai orientar o fluxo das atividades do usuário. Ela define o *desktop* (Ex.: Program Manager no Windows).

Esta barra é associada a uma janela para permitir que o usuário controle o escopo dos dados mostrados pela janela e ainda tenha uma visão geral da posição dos dados apresentados, em relação a toda base de dados disponível. As barras de rolamento da janela podem ser horizontais e/ou verticais.

Apontador

O apontador é um gráfico flutuante na tela do computador que se move em resposta às ações realizadas pelo usuário através do *mouse*. É, normalmente, por meio do apontador que o usuário seleciona os demais objetos da GUI, a fim de executar aplicações ou ativar comandos. A forma gráfica associada ao apontador pode ser alterada pelas aplicações, indicando visualmente que uma ação a ser realizada teve início ou que algum comando foi selecionado. Após o término da ação, o apontador sempre retorna à sua forma original.

Cursor

O cursor é uma linha vertical piscante que serve como um apontador de controle para a entrada de dados provenientes do teclado.

Menus

Um menu é uma lista de itens, normalmente relacionados, os quais representam os comandos e opções disponíveis para o usuário de uma aplicação. Os itens do tipo comandos são expressados por um verbo, indicando uma ação. Por exemplo, os itens "*Text*" e "*Font*" do menu na figura 2.8. Já os itens do tipo opções permitem ao usuário modificar os atributos dos dados ou propriedades da interface. Os itens "*Bold*" e "*Italic*" do menu "*Text*", na figura 2.8, são exemplos deste tipo de item. Maiores informações sobre menus podem ser encontradas na seção 3.3.5.1.

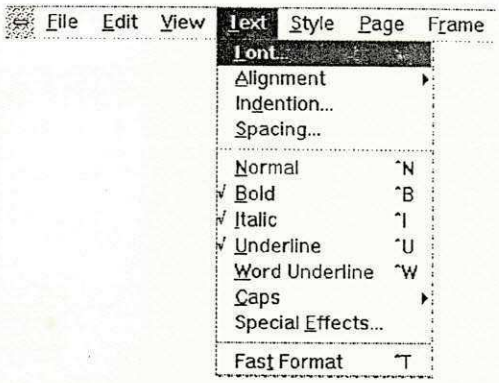

Figura 2.8 Exemplo de menu.

Controles

Os controles são objetos da interface do usuário de comportamento individual que mostram informações e aceitam dados de entrada do usuário. De uma forma geral, os seguintes controles podem ser criados em uma GUI: botões, quadros de marcar, barras de deslizamento[7], texto, quadros de lista, quadros de incremento e quadros de grupo.

• **Botões**: os controles do tipo botão se apresentam em duas formas: botões de comando e botões de rádio. Um **botão de comando** é um objeto gráfico de forma retangular, que possui um rótulo para indicar a ação associada ao botão. Os controles rótulados com *"Ok"* e *"Cancel"* e *"About"* na figura 2.9 são exemplos de botões de comandos. O rótulo de um botão de comando pode ser um texto ou símbolo gráfico. O usuário pode selecionar um botão de comando pressionando o *mouse* quando o apontador estiver sobre o botão. O **botão de rádio** é representado por um círculo seguido de um rótulo. Esse tipo de botão é usado para representar um conjunto de escolhas mutuamente exclusivas. Isto é, para cada grupo de botões de rádio o usuário pode ativar apenas uma opção por vez. Os botões rotulados com *"Monocrome"*, *"Color"*, *"Screen"* e *"Printer"* da figura 2.9 são exemplos de botões de rádio. Quando um botão de rádio é

---

[7] Neste trabalho o termo "barra de deslizamento" é utilizado como a tradução para o termo, em inglês, *slider*. Na bibliografia consultada alguns autores também utilizam o termo "controle do tipo barra de deslocamento" (*scroll bar control*) para referir-se a esse tipo de controle.

selecionado, o círculo ao lado do rótulo é preenchido.

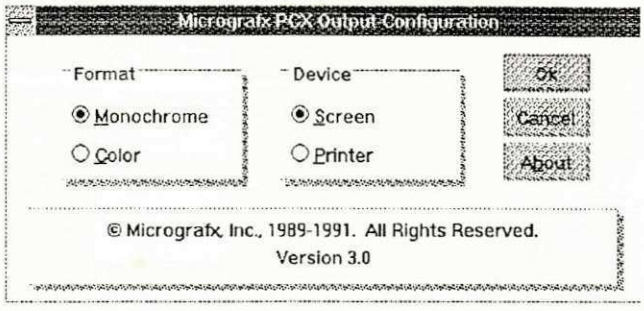

Figura 2.9 Janela com controles do tipo botões de rádio e de comando

• **Quadros de Marcar**: um **quadro de marcar** é constituído por um quadro e um rótulo (Figura 2.10). Este tipo de controle permite que o usuário ative várias opções de forma individual, mesmo que os quadros pertençam a um mesmo grupo. Um "X" dentro do quadro indica que a opção está ativa.

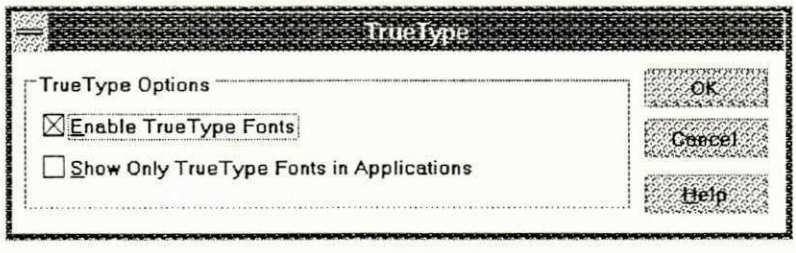

Figura 2.10 Janela com controles do tipo quadro de marcar.

• **Barras de Deslizamento**: a **barra de deslizamento** é composta por uma barra que representa uma faixa de valores e um indicador de posição. A barra de deslizamento é um tipo de controle que oferece ao usuário uma indicação visual do valor selecionado, dentro de uma faixa de valores. Esse tipo de controle é utilizado quando o programador necessita representar valores ou dimensões contínuas, como por exemplo: grau, luminosidade, volume ou velocidade. O usuário define o valor desejado deslizando o indicador de posição sobre a barra através do apontador. A figura 2.11 exemplifica duas barras de deslizamento usadas para especificar a velocidade de resposta do teclado.

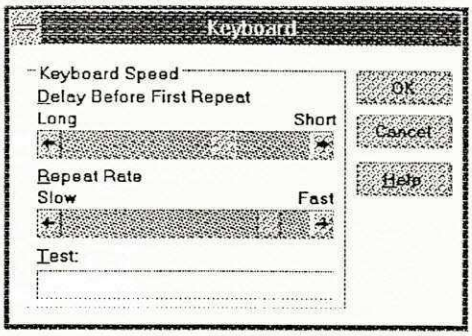

Figura 2.11 Janela com barras de deslizamento.

• **Texto**: os controles do tipo texto dividem-se em: estático e quadro editável. O **texto estático** é um rótulo ou uma imagem gráfica, usado apenas para apresentar informações que não podem ser alteradas pelo usuário (Figura 2.12). A aplicação, no entanto, pode modificar o conteúdo de um texto estático.

Figura 2.12 Janela com textos estáticos do tipo rótulo.

Já o **quadro de texto editável** é um controle composto por um rótulo e uma área editável (Figura 2.13). A área editável é usada para receber informações do usuário através do teclado. Se a área editável tiver mais de uma linha, uma barra de rolamento pode acompanhar o quadro de texto editável. Esse tipo de controle pode oferecer um texto *default* como opção para o usuário. Neste caso, o usuário pode aceitar o texto *default*, editá-lo, apagá-lo ou substituí-lo.

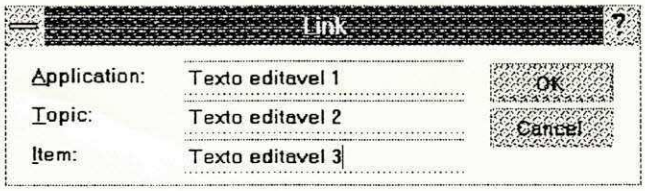

Figura 2.13 Janela com quadros de texto editável.

• **Quadros de Lista**: um **quadro de lista** é um controle composto por um rótulo e um conjunto de opções disponíveis para o usuário (Figura 2.14). Normalmente as opções de uma lista são representadas na forma de texto, no entanto, imagens gráficas também podem ser utilizadas. Quando não é possível mostrar simultaneamente todas as opções da lista dentro do quadro especificado, uma barra de rolamento vertical é geralmente associada ao quadro de lista.

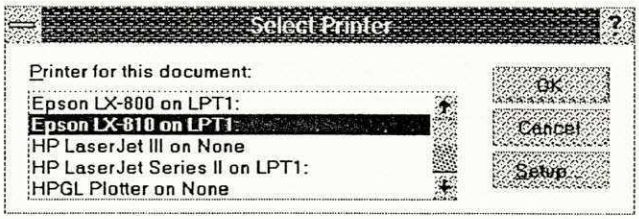

Figura 2.14 Janela com controle do tipo quadro de lista.

• **Quadros de incremento**: um **quadro de incremento** é composto por um quadro de texto editável e dois ícones de comandos. Este tipo de controle aceita apenas valores numéricos, que podem ser definidos através dos ícones de incremento ou decremento, ou através da edição direta do número via teclado na área editável (Figura 2.15).

Figura 2.15 Janela com controles do tipo quadro de incremento.

• **Quadro de Grupo**: um **quadro de grupo** é composto por um título e uma borda retangular. Esse tipo de controle não possui um comportamento individual associado a si. É apenas uma opção estética, oferecida ao programador, para facilitar a visualização de um conjunto de controles relacionados entre si. A figura 2.16 mostra uma janela com os seguintes quadros de grupo: *"pattern"*, *"applications"*, *"screen saver"*, *"wallpaper"*, *"icons"*, *"sizing grid"* e *"cursor blink rate"*.

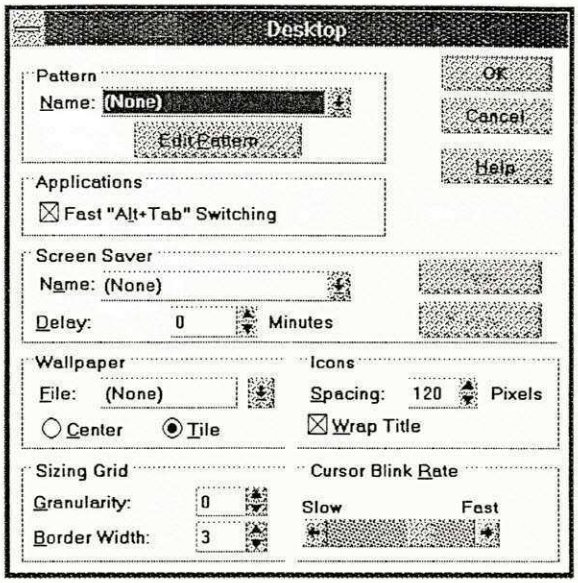

Figura 2.16 Janela com vários quadros de grupo.

Quadros de Diálogo

Os quadros de diálogos são objetos da interface do usuário compostos por uma janela e por vários controles. Os quadros de diálogo são associados a comandos que, para serem executados, necessitam de informações adicionais fornecidas pelo usuário. Por exemplo, quando o usuário ativa o comando de abertura de um arquivo em um editor de texto, informações como o nome e a localização do arquivo precisam ser fornecidas (Figura 2.17).

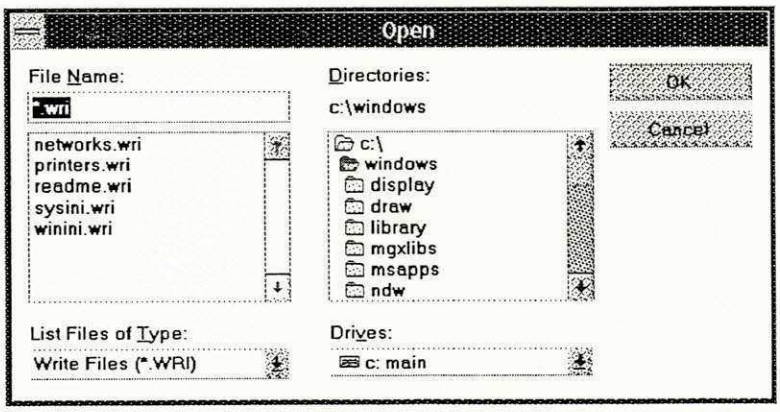

Figura 2.17 Quadro de diálogo para a abertura de arquivos.

Neste capítulo foram definidos os requisitos básicos e específicos da biblioteca UNIFIC, o próximo passo será estudar as características dos ambientes gráficos onde a programação será unificada. Esse é o assunto a ser discutido no próximo capítulo.

Capítulo 3

3. Ambientes Gráficos Unificados

Este capítulo apresenta o conjunto inicial de GUIs a ter sua programação unificada pela biblioteca UNIFIC. Inicialmente, serão definidas as GUIs cuja programação será suportada pela UNIFIC, bem como os critérios usados para a sua seleção. Em seguida, será apresentada a arquitetura de cada uma dessas GUIs, visando obter dados preliminares importantes para o processo de unificação. Finalmente, serão discutidos os aspectos mais importantes do modelo de programação das GUIs selecionadas, a fim de obter informações detalhadas para a definição do modelo de programação da UNIFIC.

3.1. Seleção das Interfaces Gráficas

Idealmente, o processo de unificação da programação entre diferentes ambientes gráficos deve envolver um estudo detalhado de um grande número de GUIs. As GUIs estudadas devem ter uma base instalada significativa, atuarem em segmentos de mercado distintos e terem sido desenvolvidas para sistemas operacionais diferentes. Um subconjunto adequado de GUIs para a realização desse estudo poderia ser: Windows, OSF/Motif, OS2/Presentation Manager, OPEN LOOK e Macintosh. Entretanto, um estudo amplo sobre cada uma dessas GUIs consumiria um tempo considerável, inviabilizando a realização deste trabalho em um período de tempo aceitável. Para contornar este problema, tornou-se necessário definir um subconjunto mínimo de GUIs a partir do qual fosse possível obter um modelo de programação unificado genérico. Foram

selecionados dois ambientes gráficos distintos com características bem particulares : Windows e Motif. A seguir será feita uma descrição sucinta de cada uma destas GUIs e serão apresentados os motivos que levaram a esta escolha.

3.1.1. Windows

O Windows é um sistema de janelas multitarefa, desenvolvido pela Microsoft, para oferecer uma GUI aos usuários de PCs baseados em MS-DOS e seus compatíveis. O fato de o sistema Windows ter sido desenvolvido para um ambiente monousuário, exige que as aplicações residam, sejam executadas e tenham sua saída mostrada na mesma máquina onde o Windows encontra-se instalado (Figura 3.1).

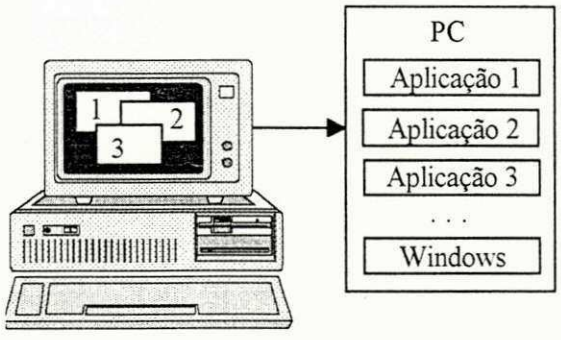

Figura 3.1 Visão de várias aplicações sendo executadas com o Windows em um PC.

Os motivos que levaram a GUI Windows a ser uma das GUIs selecionadas para o processo de unificação foram: tamanho da base instalada e aumento do número de plataformas compatíveis.

• **Tamanho da Base Instalada**: o Windows é uma GUI que ocupa posição de destaque no mercado das interfaces gráficas. Em outubro de 1992, 38% das aplicações disponíveis para PCs no mercado mundial eram aplicações Windows.

• **Aumento do Número de Plataformas Compatíveis**: apesar do Windows ter sido originalmente desenvolvido para um ambiente monousuário e proprietário, esforços estão sendo desenvolvidos para estender a compatibilidade das aplicações Windows para outros ambientes. Os

dois primeiros esforços desenvolvidos neste sentido foram: o lançamento de uma versão do Windows para redes locais e o desenvolvimento de ABIs, ou *Application Binary Interfaces* para Windows. A versão do Windows para redes locais, denominada *Windows for Workgroup*, acrescenta ao Windows um módulo para a gerência de LAN[8] e assegura a compatibilidade das aplicações com os padrões de conectividade NetWare. Apesar do *Windows for Workgroup* permitir que aplicações Windows compartilhem recursos como discos rígidos e impressoras, o nível de compatibilidade exigido entre as máquinas conectadas à rede - IBM PCs ou Macintosh - ainda é muito alto. A fim de resolver este problema, a solução mais recentemente adotada foi o desenvolvimento de ABIs para Windows. As ABIs oferecem portabilidade binária de código entre máquinas distintas. Por exemplo, a ABI Wabi, disponível comercialmente, permite a execução de aplicações Windows em ambientes UNIX. Se o número de bases instaladas do Windows já era expressivo, com o aumento das perspectivas de compatibilidade para novos ambientes este número só tende a aumentar.

3.1.2. Motif

O Motif é um sistema desenvolvido pela OSF[9] que oferece uma GUI para plataformas de *hardware* que utilizam o sistema operacional UNIX. O Motif baseia-se em uma filosofia de rede. Isto permite que aplicações, cujas saídas são apresentadas em janelas distintas da tela de uma mesma estação de trabalho, possam estar sendo executadas em diferentes máquinas remotas, conectadas através de uma rede de comunicação (Figura 3.2).

Os motivos que levaram o Motif ser o segundo ambiente gráfico a ser unificado pela

---

[8] LAN, ou Local Area Network, é um termo usado para designar um grupo de computadores e outros dispositivos, dispersos por uma área relativamente limitada (um departamento ou prédio comercial, por exemplo), conectados entre si através de um canal de comunicação. O canal de comunicação permite que os dispositivos interajam uns com os outros.

[9] A OSF, ou Open Software Foundation, é uma Organização formada por empresas como a Hewlett-Packard, Digital, IBM, e dezenas de outras corporações, com a finalidade de desenvolver especificações de *software* e tecnologia para sistemas abertos.

biblioteca UNIFIC foram: compatibilidade em várias plataformas de *hardware* e tendência a padrão *de facto*.

• **Compatibilidade em Várias Plataformas de Hardware**: o Motif permite que suas aplicações sejam executadas compativelmente em um grande número de plataformas. Esta compatibilidade estende-se desde computadores simples, como PCs, até máquinas mais sofisticadas, como mini ou supercomputadores.

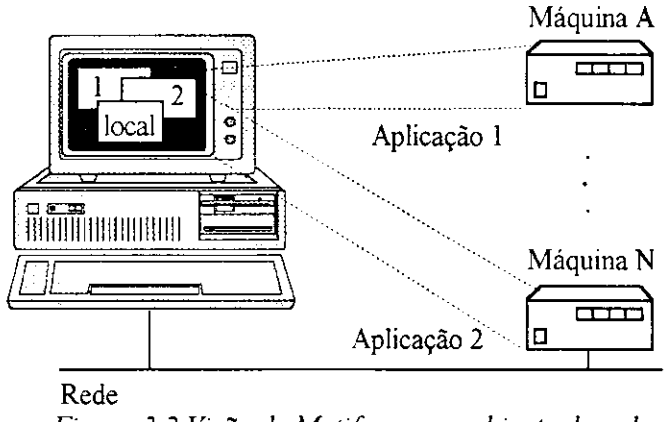

Figura 3.2 Visão do Motif em um ambiente de rede.

• **Tendência a Padrão de *Facto***: o Motif é o mais forte candidato a padrão *de facto* para sistemas abertos[10], uma vez que recentemente um grupo de grandes empresas no segmento UNIX, incluindo a HP, The Santa Cruz Operation, a Sun, a Novel, e até mesmo a IBM, anunciaram que adotarão a GUI Motif como o seu sistema de janelas padrão.

Apesar de apenas as GUIs Windows e Motif terem sido selecionadas como base para o processo de unificação, as discussões apresentadas neste trabalho serão úteis para a implementação da UNIFIC em qualquer ambiente gráfico. Para as GUIs que se baseiam na filosofia do Windows ou do Motif, o modelo de programação poderá ser implementado diretamente. Nessa classe encontram-se a maioria das GUIs do mercado. Este é o caso de ambientes como OPEN LOOK ou Presentation Manager, por exemplo. Para as demais GUIs, se o

---

[10] Sistemas abertos, ou *Open Systems*, são sistemas, normalmente multiusuários, que adotam padrões de portabilidade, escalabilidade e interoperabilidade, aceitos internacionalmente.

modelo de programação não for diretamente implementável, as discussões a respeito do processo de unificação serão úteis para direcionar o estudo da nova GUI e adequar o modelo de programação unificado à nova filosofia.

3.2. Apresentação das Arquiteturas

Um estudo mais específico da infra-estrutura utilizada por cada uma das GUIs selecionadas torna-se necessário, para que seja possível definir um modelo de programação unificado genérico. As arquiteturas das GUIs Windows e Motif serão apresentadas a seguir.

3.2.1. Arquitetura Windows

A arquitetura do Windows é composta por um conjunto de serviços construídos sobre o sistema operacional MS-DOS. Os principais componentes da arquitetura Windows são: o núcleo, as interfaces de dispositivos gráficos (Grafical Device Interface ou GDI) e o módulo do usuário (Figura 3.3).

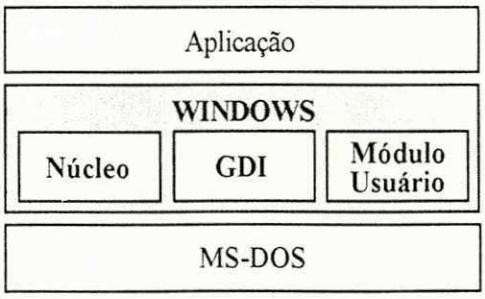

Figura 3.3 Módulos componentes da arquitetura Windows.

3.2.1.1. Núcleo

O núcleo do Windows é o módulo responsável pela realização dos serviços a nível de sistema operacional. Este módulo implementa alguns serviços complementares ao MS-DOS, para viabilizar o funcionamento multitarefa do Windows. O único serviço que continua a ser

totalmente realizado pelo MS-DOS, sem interferência do Windows, é a entrada e saída em arquivos.

Os principais componentes do núcleo são: gerente de recursos do sistema, gerente de memória virtual e escalonador de aplicações (Figura 3.4).

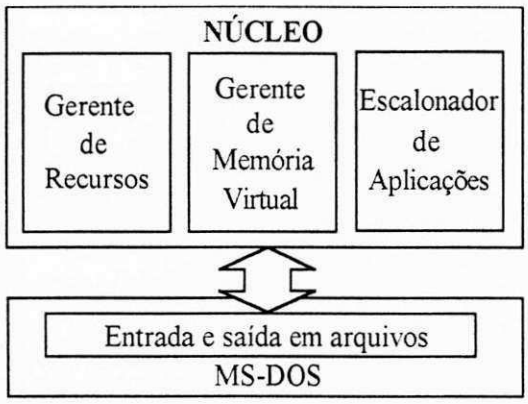

Figura 3.4 Principais componentes do núcleo do Windows e sua interação com o DOS.

• **Gerente de recursos**: é o responsável pela coordenação do acesso das aplicações aos recursos compartilhados no sistema. Alguns exemplos destes recursos são teclado, *mouse*, tela, impressoras e portas seriais.

• **Gerente de memória virtual**: é o responsável pelo mapeamento dos segmentos de memória entre a memória física e o disco. Este gerente é um dos componentes do Windows que aperfeiçoa o esquema de gerência de memória do DOS, viabilizando o desenvolvimento de um sistema de janelas multitarefa neste ambiente.

• **Escalonador de aplicações**: define, dentre as várias aplicações que estão sendo executadas no Windows, qual aplicação terá acesso à CPU.

3.2.1.2. Interface de Dispositivos Gráficos

A interface de dispositivos gráficos, ou *Graphic Device Interface* (GDI) é o módulo que serve de interface entre as aplicações e os dispositivos gráficos disponíveis no sistema. Os

principais componentes da GDI são: *os drivers* de dispositivos, o gerente de atributos gráficos e uma biblioteca gráfica (Figura 3.5).

• ***Drivers* de dispositivo**: são os *drivers* que permitem às aplicações Windows realizarem operações gráficas de forma independente de dispositivos.

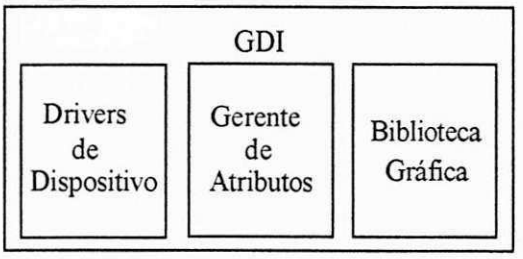

Figura 3.5 Principais compomentes do Módulo GDI.

• **Gerente de atributos**: coordena a definição e o acesso das aplicações aos atributos utilizados em uma operação gráfica.

• **Biblioteca gráfica**: é um conjunto de funções através das quais os atributos do gerente de atributos e os *drivers* de dispositivos podem ser acessados pela aplicação, para a geração de saída gráfica.

3.2.1.3. Módulo do Usuário

O módulo do usuário oferece todos os demais serviços disponíveis na GUI Windows. Esse módulo é responsável, basicamente, pela criação e gerência dos objetos da GUI, inclusive pela interação entre os objetos da GUI e o usuário. Os componentes que mais se destacam no módulo do usuário são: biblioteca de objetos da GUI, gerente de janelas e gerente de eventos (Figura 3.6). Esse módulo normalmente interage com o GDI para mostrar, na tela, os objetos criados pela aplicação.

• **Biblioteca de objetos da GUI**: define a aparência e o comportamento *default* de todos os objetos da GUI.

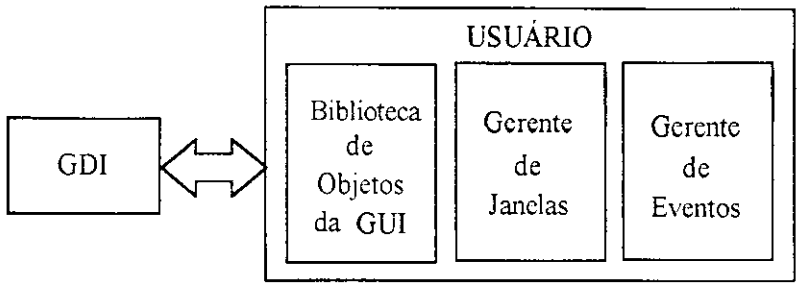

Figura 3.6 Principais componentes do módulo usuário e sua interação com o GDI.

• **Gerente de janelas**: coordena as características visuais e comportamentais das janelas ativas no sistema. Controla, por exemplo, qual janela recebe a entrada do usuário e como as janelas mudam de tamanho.

• **Gerente de eventos**: recebe os eventos gerados pelo *hardware* de entrada, pelas aplicações ou pelo próprio sistema e os envia para a aplicação destino.

3.2.2. Arquitetura Motif

A arquitetura do Motif é composta por duas camadas: X Window e componentes Motif (Figura 3.7).

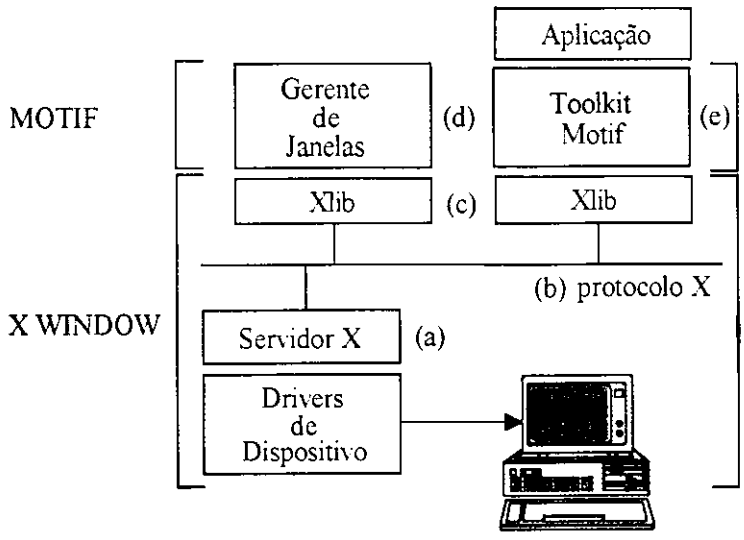

Figura 3.7 Principais componentes da arquitetura Motif.

O X Window é o sistema de janelas que forma a camada básica do Motif. Os componentes

do sistema X Window são: Xlib, protocolo X e servidor X. Sobre o X Window é que se encontra o Motif propriamente dito. Os componenetes particulares do Motif são, basicamente, um gerente de janelas e um *toolkit*. Cada um dos elementos da arquitetura Motif será analisado detalhadamente a seguir:

3.2.2.1. Sistema X Window

O X Window é um sistema de janelas gráfico, multitarefa, distribuído, independente de dispositivos, desenvolvido para permitir a execução de aplicações com GUIs em ambientes de redes heterogêneas. A fim de permitir que as aplicações remotas acessem o terminal de saída, esse sistema de janelas baseia-se no modelo cliente-servidor. Nesse esquema, uma aplicação cliente (cliente X), que pode estar sendo executada remotamente, faz pedidos a um servidor gráfico (servidor X) para acessar o seu terminal de saída. Os componentes do X Window são: servidor X, protocolo X e o Xlib.

• **Servidor X** (Figura 3.7a): é um processo servidor que deve ser executado na estação de trabalho que possui um terminal gráfico. As principais atribuições deste servidor são: permitir que vários clientes acessem o mesmo vídeo, monitorar os eventos gerados pelos usuário e os enviar para a aplicação adequada, além de atender aos pedidos de saída gráfica, feitos pelas aplicações clientes.

• **Protocolo X** (Figura 3.7b): é o protocolo utilizado para a comunicação entre o servidor X e os clientes X.

• **Xlib** (Figura 3.7c): é uma biblioteca de baixo nível, escrita em linguagem "C", para dar suporte aos serviços oferecidos pelo protocolo X. Esta biblioteca oferece um conjunto de rotinas básicas que permitem, por exemplo, estabelecer uma conexão com um servidor em particular, enviar e receber eventos através da rede, desenhar gráficos e criar os objetos da GUI.

3.2.2.2. Gerente de Janelas

O gerente de janelas do Motif (figura 3.7d) é o módulo responsável pela realização das funções que devem ser consistentes para todas as janelas do sistema, como por exemplo, mover, iconizar ou mudar o tamanho de uma janela. Também é o gerente de janelas que determina qual aplicação recebe a entrada do usuário.

3.2.2.3. Toolkit Motif

O *toolkit* Motif (figura 3.7e) implementa o *look-and-feel* dos objetos da GUI Motif, oferecendo funções para a criação e gerência destes objetos, além de funções de mais alto nível do que a Xlib para o envio e recebimento dos eventos.

As principais características dos ambientes gráficos, nos quais a biblioteca UNIFIC estará disponível, foram apresentadas. A seguir será estudado o modelo de programação adotado por cada ambiente; esse estudo possibilitará a elaboração do modelo unificado apresentado no capítulo 4.

3.3. Modelo de Programação das GUIs Unificadas

Nesta seção, será feito um estudo dos modelos de programação das GUIs Windows e Motif. Esse estudo, baseado nos requisitos específicos da UNIFIC apresentados na seção 2.3, oferece subsídios para a definição do modelo de programação da UNIFIC. A seguir serão apresentados os principais aspectos dos modelos de programação do Windows e do Motif os quais se relacionam com as seguintes operações: serviços do sistema operacional, troca de dados entre aplicações, entrada de dados, saída gráfica e objetos da GUI.

3.3.1. Modelo de Serviços do Sistema Operacional

O modelo de serviços do sistema operacional do Windows e do Motif pode ser classificado em quatro grupos de operações: execução de aplicações, gerenciamento de memória dinâmica, entrada e saída em arquivos e manipulação de cadeias.

3.3.1.1. Execução de Aplicações

Tanto no Windows quanto no Motif existem meios de uma aplicação criar um processo para executar uma outra aplicação. No Windows existem funções pré-definidas, especificamente implementadas pela GUI, que permitem a uma aplicação ativar a execução de aplicações DOS ou Windows.

Já no Motif, não existem funções específicas que realizem diretamente esse tipo de operação. Ao invés disso, as próprias funções disponíveis na linguagem "C" para esse propósito, como as chamadas ao sistema *system()*, *popen()*, *fork()* ou *exec()*, são oferecidas ao programador. O problema que ocorre quando o programador executa uma aplicação a partir de outra usando apenas esse tipo de função, é que não há meios do processo pai detectar os erros ocorridos durante a execução da aplicação filho. Para que isso seja possível, é necessário que o programador projete o seu próprio esquema de recuperação de erros. Nesse caso, o programador deve tratar várias condições de erro durante a execução da outra aplicação que incluem, desde a impossibilidade de se criar o processo filho até a ocorrência de erros no protocolo de comunicação. Maiores informações sobre a implementação, no ambiente Motif, de um esquema para tratar erros ocorridos durante a execução de processos filhos, podem ser encontrados na referência [Heller91].

3.3.1.2. Gerenciamento de Memória Dinâmica

Existem diferenças significativas entre o esquema de gerenciamento de memória no Windows e no Motif. Estas diferenças devem-se às características dos ambientes para os quais cada GUI foi implementada. Enquanto no Motif existe ampla disponibilidade de recursos, no Windows eles estão sujeitos a limitações impostas pelo MS-DOS.

Para contornar os limites de memória impostos pelo MS-DOS e oferecer um sistema multitarefa, o Windows implementa um esquema particular de gerência de memória. Nesse esquema, para que o programador aloque uma área de memória dinamicamente, é necessário que ele defina o tipo de memória que deseja alocar e os atributos da área alocada.

Dois **tipos** de memória podem ser alocados dinamicamente no Windows: local ou global. Quando uma aplicação faz uma **alocação local**, a área de memória é alocada no segmento de dados *default* da aplicação. O segmento de dados *default* de uma aplicação Windows é dividido em: dados estáticos, pilha e *heap* local (figura 3.8). O *heap* local é a área disponível para alocação local em uma aplicação Windows.

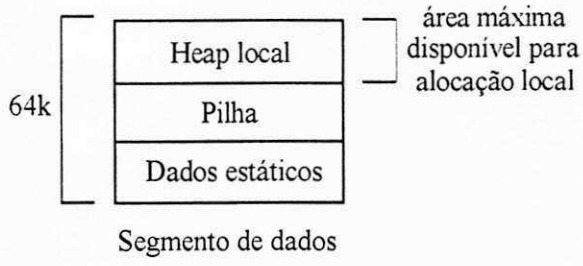

Figura 3.8 Componentes do segmento de dados default de uma aplicação Windows.

A vantagem desse tipo de alocação é que ele não causa *overhead*. A desvantagem é que esta alocação limita a memória alocada dinamicamente a um tamanho máximo de 64k[11] e uma área de memória alocada localmente não pode ser compartilhada entre aplicações. A primeira

[11] Na prática, a área disponível para alocação de memória local no Windows é menor do que 64k, uma vez que também é no segmento *default* de dados onde se alocam a pilha e os dados estáticos da aplicação.

desvantagem deve-se ao tamanho de um segmento no ambiente Windows, que é de 64k e a segunda desvantagem deve-se ao fato de que uma área alocada localmente não pode ser vista por outras aplicações.

A **alocação global**, por sua vez, não apresenta limites de tamanho, pois, para este tipo de alocação, a área é obtida da memória global do Windows. A memória global é toda área de memória controlada pelo Windows (Figura 3.9). Esta área começa onde o MS-DOS carrega inicialmente o Windows na memória e estende-se até o final da área de memória disponível no computador, que corresponde, geralmente, ao topo da área física disponível na máquina[12]. A única desvantagem desse tipo de alocação é o seu alto *overhead*. Esse *overhead* deve-se ao esquema de gerenciamento de memória multisegmento adotado pelo Windows. Quando o programador aloca uma área na memória global, ele está alocando um segmento. Internamente, além da área requisitada, o Windows aloca mais 24 *bytes* de memória - 16 *bytes* para a ligação entre os segmentos na memória e 8 *bytes* para a entrada na tabela de segmentos.

No Win32[13], apesar de um esquema de memória virtual estar disponível e os segmentos na memória não serem visíveis à aplicação, tanto a alocação local quanto a global continuam disponíveis para manter a compatibilidade com as versões anteriores do Windows. Entretanto, nesse caso, a alocação local não estará mais limitada a uma área de 64k. O Win32 permite que o programador aloque, facilmente, vários *heaps* locais e aumente assim a área disponível para alocação local[14]. Maiores detalhes sobre esse assunto podem ser encontrados na referência [Yao91].

---

12 A área de memória disponível no Windows será maior do que a área física disponível na máquina, apenas quando o Windows for executado no modo 386-Melhorado. Quando executado nesse modo, o Windows utiliza um esquema de memória virtual.

13 O Win32 é uma extensão do Windows projetada para dar suporte a características avançadas como: suporte para múltiplos processadores, processamento distribuído, ambiente de rede e segurança.

14 Este tipo de operação também pode ser realizado no Win16. Entretanto, nesse caso existem duas desvantagens. Primeiro, o programador terá que programar em baixo nível utilizando linguagem assembler. Segundo, a documentação disponível sobre esse assunto é escassa.

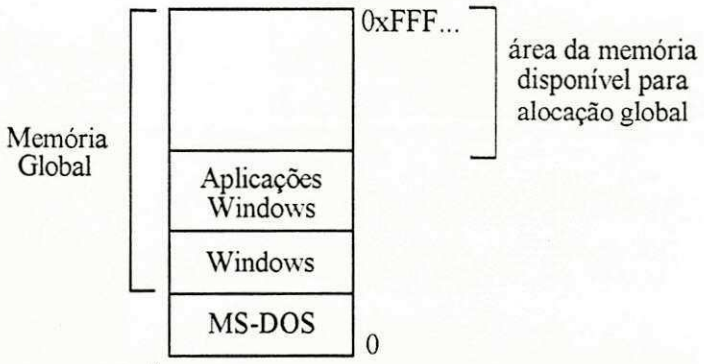

Figura 3.9 Área disponível para alocação global no Windows

Dois tipos de **atributos** podem ser especificados pelo programador durante a alocação de memória local ou global no Windows: móvel ou fixa. As áreas móveis ainda podem ser definidas como descartáveis.

As áreas alocadas como **móveis** podem ser relocadas na memória, a critério do gerente de memória, ou paginadas para disco quando o sistema precisar de memória. Do ponto de vista do programador, sempre que for necessário acessar uma área móvel, esta área deve ser previamente travada. Isto é necessário devido ao modelo de endereçamento de memória utilizado pelo MS-DOS. Os *handles* de acesso a áreas de memória em uma aplicação executada em ambiente DOS referenciam, indiretamente, blocos na memória física. Portanto, essas aplicações possuem uma visão direta do espaço de endereçamento físico da máquina onde estão sendo executadas. É de responsabilidade do programador garantir que os dados desejados estejam presentes na memória física quando forem acessados. No Windows essa garantia é obtida através do travamento prévio das áreas de memória a serem acessadas.

As áreas **móveis e descartáveis** podem ser simplesmente descartadas da memória quando o sistema precisar de memória. Este tipo de área de memória deve ser definido apenas para dados que não mudam durante a execução da aplicação e por isto podem ser recarregados sempre que for necessário.

As áreas de memória **fixas** não podem ser paginadas para disco nem relocadas pelo sistema. Um grande volume de memória fixa pode prejudicar o funcionamento do esquema de gerenciamento de memória do Windows. Esse atributo deve ser usado com moderação.

Devido ao esquema de gerência de memória explicado acima, além da função que permite a alocação dinâmica de memória, a API Windows coloca à disposição do programador funções para travamento, destravamento, relocação e liberação de áreas de memória. Durante a alocação de uma área de memória global, o programador pode indicar se a área será compartilhada. As áreas de memória compartilhadas no Windows, entretanto, só estarão disponíveis para troca dinâmica de dados (vide seção 3.3.2.2). Maiores detalhes sobre a gerência de memória do Windows podem ser encontrados nas referências [Mcsft90b] e [Petzold90].

No Motif, o tipo de memória disponível para alocação dinâmica é o mesmo do sistema operacional UNIX. Isto é, toda memória alocada dinamicamente é sempre **global** e **fixa**. O tipo de memória é sempre global porque o modelo de endereçamento de memória adotado pelo UNIX esconde do programador o conceito de segmento. Logo, não existe o conceito de memória local. Já o atributo é fixo porque, no UNIX, as aplicações utilizam apenas endereços lógicos de memória. Como o mapeamento dos endereços lógicos para endereços físicos é feito pela MMU[15], torna-se transparente para a aplicação qualquer mudança de localização que venha a ocorrer nos blocos de dados da memória física.

Na biblioteca de programação do Motif estão à disposição do programador funções para a alocação, relocação e liberacão de áreas de memória.

---

[15] MMU ou Memory Management Unit é uma unidade de *hardware* responsável, entre outras coisas, pelo mapeamento dos endereços lógicos utilizados por uma aplicação para os endereços físicos na memória, garantindo, com a ajuda do sistema operacional, que a informação a ser acessada esteja disponível na memória física quando requisitada.

3.3.1.3. Entrada e Saída em Arquivos

O Windows e o Motif oferecem um conjunto de funções para realizar entrada e saída de dados em arquivos, em conformidade com os seus respectivos sistemas operacionais. O modelo de programação utilizado por uma aplicação que deseja acessar um arquivo nesses ambientes é bem semelhante. Antes que qualquer operação seja realizada no arquivo desejado, é necessário que ele esteja aberto. Em seguida, qualquer operação de leitura ou escrita pode ser realizada. Outras facilidades como *seek* ou *append* em arquivos também estão disponíveis. Quando não for mais necessário acessar o arquivo, ele deverá ser fechado.

3.3.1.4. Manipulação de Cadeias

A manipulação de cadeias varia muito entre o Windows e o Motif. No Windows, as cadeias podem ser definidas como dados no arquivo fonte ou como recursos no arquivo de recursos[16]. As cadeias definidas diretamente no código fonte da aplicação são vistas como variáveis convencionais e manipuladas através de funções de concatenação, cópia ou comparação com outras cadeias, de forma semelhante a manipulação de cadeias disponível na linguagem "C", por exemplo. Já as cadeias definidas como recursos são criadas através de comandos específicos no arquivo de recursos e só são carregadas pela aplicação quando necessário. Essa forma de definir uma cadeia permite maior facilidade para tradução de aplicações para outras línguas (internacionalização), uma vez que todas as cadeias ficam concentradas em um único arquivo.

No Motif, para que as cadeias possam ser manipuladas pelos objetos da GUI, elas devem ser convertidas para um formato padrão interno, definido pelo próprio Motif. Este formato é representado pela tripla: conjunto de caracteres particular, direção e texto. Este tipo de

---

[16] O arquivo de recursos é um arquivo que contém especificações sobre os objetos da GUI que podem ser armazenados como recursos. Os recursos são dados *read-only* que são carregados pela aplicação na medida que forem necessários. Alguns exemplos de dados armazenados como recursos no Windows são: *bitmaps*, cursores, ícones, fontes e cadeias.

manipulação de cadeias permite que se represente, com facilidade, alfabetos estrangeiros cujo conjunto de símbolos não pode ser representado apenas por uma variável do tipo *char* ou cujo sentido da escrita seja da direita para esquerda, como por exemplo, o hebreu.

3.3.2. Modelo de Troca de Dados

O segundo modelo a ser analisado é o modelo de troca de dados entre aplicações. Esta seção tem o objetivo de discutir a disponibilidade dos mecanismos de *clipboard* e de troca de dados através de mensagens nas duas GUIs. Além de apresentar um comparativo entre o modelo de programação adotado por cada GUI para a utilização destes mecanismos.

3.3.2.1. Clipboard

O mecanismo de troca de dados através do *clipboard* está disponível no Windows. Para trocar dados através deste mecanismo, o programador deve utilizar funções especificamente definidas para este fim. Durante a cópia de dados para o *clipboard* a aplicação deve abrir o *clipboard*, esvaziá-lo e colocar os dados desejados. Vários formatos de dados podem ser trocados através do *clipboard*. Desde formatos pré-definidos, como por exemplo, texto ou *bitmap*, até formatos definidos pela própria aplicação. No caso da obtenção dos dados do *clipboard*, a aplicação deve abri-lo, podendo consultar os tipos de dados disponíveis, copiá-los e, então, fechar o *clipboard*.

O *clipboard* é o único mecanismo de troca de dados entre aplicações, oferecido pela GUI Motif. A troca de dados através do *clipboard* no Motif é feita de forma semelhante à do Windows. Para enviar dados ao *clipboard*, a aplicação deve inicializar o processo de transferência, registrar o tipo do dado que será armazenado, copiar os dados e finalizar o processo de cópia, através de funções específicas. O Motif também oferece alguns formatos de

dados pré-definidos, como a cadeia e o *bitmap,* e ainda permite que a aplicação registre qualquer formato de dados em particular. Já para se obter os dados do *clipboard*, uma simples função de leitura pode ser usada. Funções para a obtenção do tipo e tamanho dos dados armazenados no *clipboard* também estão disponíveis.

3.3.2.2. Troca de Mensagens

A troca de dados entre aplicações que se baseia no mecanismo envio e rebebimento de mensagens no Windows, é conhecido como troca dinâmica de dados. A troca dinâmica de dados utiliza o próprio sistema de eventos da GUI para implementar o envio e o recebimento das mensagens entre as aplicações Windows. Nesse esquema, quando uma aplicação cliente deseja obter algum dado de outra, ela envia um *broadcast* para descobrir que aplicação servidora pode fornecê-los. A aplicação servidora que responde ao *broadcast* estabelece uma conversação com a aplicação cliente, para enviar os dados desejados.

A troca de dados entre aplicações realizadas através deste mecanismo no Windows pode ser de três tipos: fria, quente ou morna. No caso de uma **troca fria**, depois que os dados são fornecidos à aplicação cliente, ela perde o contato com a aplicação servidora e, se o valor dos dados fornecidos forem alterados no servidor, o cliente não toma conhecimento. No caso de uma **troca quente**, sempre que houver mudança nos dados no servidor, este enviará os dados atualizados aos clientes que tiverem consultado aquela informação. No caso de uma **troca morna**, o servidor informa ao cliente apenas que houve mudança nos dados fornecidos e, se for de interesse do cliente, ele pedirá, explicitamente, os novos dados ao servidor.

A troca de dados através do esquema de troca de mensagens pode ser implementado pelo programador de aplicações Motif apenas para a troca de dados entre um grupo pré-determinado de aplicações, mas a GUI não oferece nenhum suporte especial para isto. O mecanismo há de ser

implementado a nível de X Window, utilizando-se as funções da Xlib e realizando-se a troca de dados através do servidor X que serve como intermediário e como repositório temporário para os dados compartilhados entre as aplicações. Um esquema simplificado deste mecanismo é apresentado na figura 3.10.

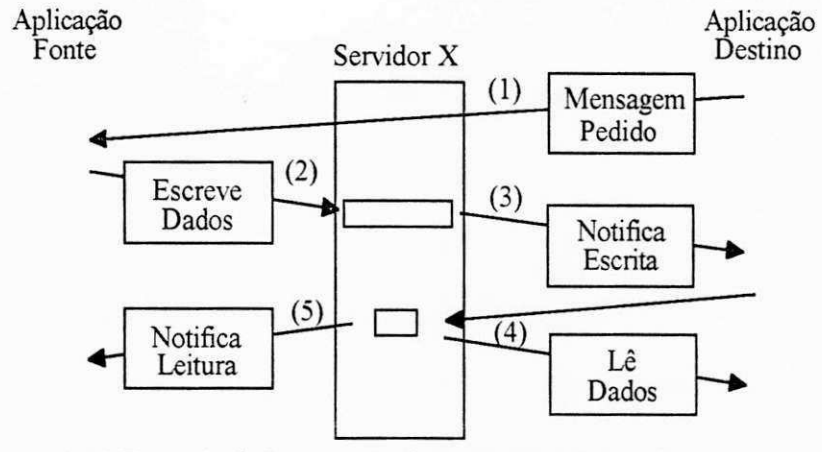

Figura 3.10 Troca de dados através do esquema de troca de mensagens no Motif

(1) A aplicação destino envia uma mensagem para a aplicação fonte, informando os dados que deseja obter (a aplicação destino deve conhecer o formato e a localização dos dados desejados).

(2) A aplicação fonte escreve os dados pedidos no servidor X.

(3) O servidor X informa à aplicação destino que os dados desejados estão disponíveis.

(4) A aplicação destino lê os dados armazenados no servidor X.

(5) A aplicação destino é notificada pelo servidor X que os dados já foram lidos.

A utilização do servidor X como um repositório temporário de dados, no entanto, não é um processo otimizado no X Window, e por isto, sua utilização somente será viável para a troca de pequenas quantidades de dados. Informações detalhadas sobre a utilização do mecanismo de troca de mensagens no Motif para a troca de dados entre aplicações podem ser encontradas nas referências [Berlage91], [Keller90] e [Nye90].

3.3.3. Modelo de Entrada de Dados Orientado a Eventos

Para facilitar o estudo do modelo de entrada de dados orientado a eventos do Windows e do Motif, ele será dividido em três fases: geração, recebimento e processamento do evento.

3.3.3.1. Geração dos Eventos

Os eventos no Windows podem se originar de duas fontes: *hardware* ou *software*. Os eventos que têm origem do *hardware* (*mouse* ou teclado) são gerados pelos respectivos *drivers* e colocados na fila de eventos do sistema. Em seguida, o próprio Windows encarrega-se de enviar os eventos da fila de eventos do sistema, para a fila de eventos da aplicação à qual pertencem. Os eventos que têm origem de *software*, por outro lado, são os eventos gerados pelas próprias aplicações ou pelo sistema Windows, através de chamadas às funções próprias para este fim. Este tipo de evento é enviado diretamente para a função de processamento, associada à janela que recebeu o evento, sem passar por filas. Maiores informações sobre a geração de eventos no Windows podem ser encontradas na referência [Norton90].

No Motif, as fontes de origem dos eventos também são o *hardware* e o *software*. Da mesma forma que no Windows, a interação do usuário com o *mouse* ou teclado dá origem aos eventos de *hardware*. No Motif, no entanto, é o servidor X que, após a detecção da interação entre o usuário e o *hardware* de entrada, determina o tipo de evento que deve ser gerado e o coloca na fila de eventos do sistema. Tambem é o servidor X que retira os eventos da fila de eventos do sistema e os envia para a fila de eventos da aplicação adequada. Já os eventos que têm origem de *software* são gerados pelas aplicações, quando precisam se comunicar entre si, ou pelo servidor X encarregado de gerar eventos, quando os julga necessários. Independentemente da origem, o envio e recebimento de eventos no Motif sempre é coordenado pelo servidor X. Maiores informações sobre a geração de eventos no Motif podem ser encontradas nas referências

[Berlage91], [Nye90] e [Heller91].

3.3.3.2. Recebimento dos Eventos

As aplicações Windows utilizam um sistema de laço infinito para obter os eventos da fila de eventos da aplicação. Este laço deve ser construído pelo programador Windows a partir das funções da biblioteca UNIFIC, para, no mínimo: ler os eventos da fila de eventos da aplicação, interpretá-los e enviá-los para a função de processamento adequada.

No Motif, um esquema semelhante é utilizado. A única diferença é que a biblioteca de programação Motif já oferece ao programador uma função única que implementa toda funcionalidade necessária a um laço de recebimento de eventos para aplicações Motif.

3.3.3.3. Processamento dos Eventos

A filosofia de processamento de eventos adotada pelo Windows difere bastante da adotada pelo Motif. No primeiro, a aplicação obtém os eventos da fila de eventos da aplicação e os envia para a janela a qual foram destinados (Figura 3.11a).

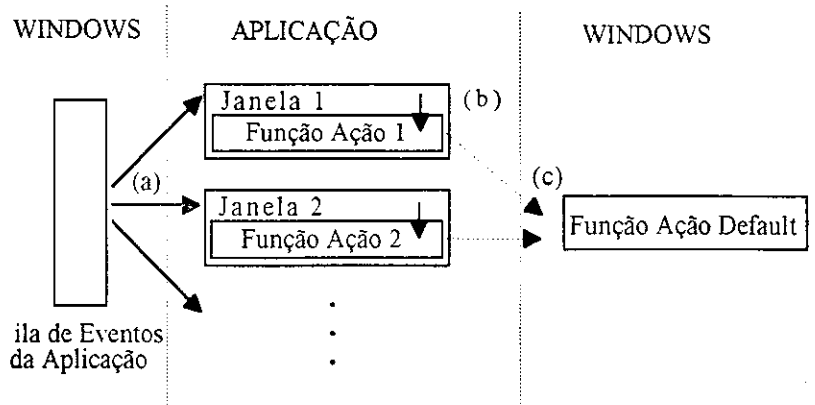

Figura 3.11 Esquema de processamento de eventos no Windows

Após receber um evento, a janela o envia para ser processado pela sua função de processamento (figura 3.11b). Cada janela principal ou secundária no Windows, possui uma função de processamento associada a si denominada **função-ação**. A função-ação define o

conjunto de eventos de interesse da janela e a ação a ser executada pela aplicação quando algum destes eventos for recebido. No Windows, a função-ação de uma janela deve incluir também os eventos de interesse dos objetos que compõem aquela janela. Os eventos de interesse da barra de menu de uma janela, por exemplo, devem ser especificados na função-ação da janela à qual o menu pertence. Quando o evento recebido pela janela não for processado pela função-ação da janela, ele será enviado para a função-ação *default* do Windows (figura 3.11c). A função-ação *default* é uma função, oferecida pela API Windows, que define qual deve ser o comportamento *default* das aplicações para os eventos ocorridos no ambiente Windows.

No Motif, quando um evento é obtido da fila de eventos da aplicação, ele é enviado diretamente para o objeto associado, seja ele uma janela, um menu ou um controle (figura 3.12a). Depois do objeto receber o evento, ele poderá processá-lo de duas formas: ação *default* do *toolkit* ou função-ação da aplicação. O *toolkit* Motif define, para cada um de seus objetos, o conjunto de eventos que pode ser recebido por esse objeto. Para cada um desses eventos, o *toolkit* oferece uma ação *default*. Se o evento, recebido pelo objeto, modificar apenas a aparência deste objeto, este evento será processado diretamente pela função-ação *default*, oferecida pelo *toolkit* (figura 3.12b). Se, por outro lado, o evento recebido ativar a funcionalidade da aplicação, será necessário verificar se o programador associou uma função-ação a este evento. Se a função-ação foi registrada, o endereço desta função é obtido e a função correspondente é chamada (figura 3.12c). Se o programador não registrar nenhuma função-ação para o evento, então, o evento também será processado de forma *default* pelo *toolkit* Motif. É importante observar que, enquanto no Windows as funções-ação são associadas às janelas, no Motif as funções-ação são associadas aos eventos, de forma particular para cada objeto da GUI. Geralmente, mais de uma função-ação é registrada no Motif para um determinado objeto, uma para cada evento de interesse do objeto.

Maiores informações sobre o esquema de processamento de eventos do Motif podem ser encontradas na referência [Heller91].

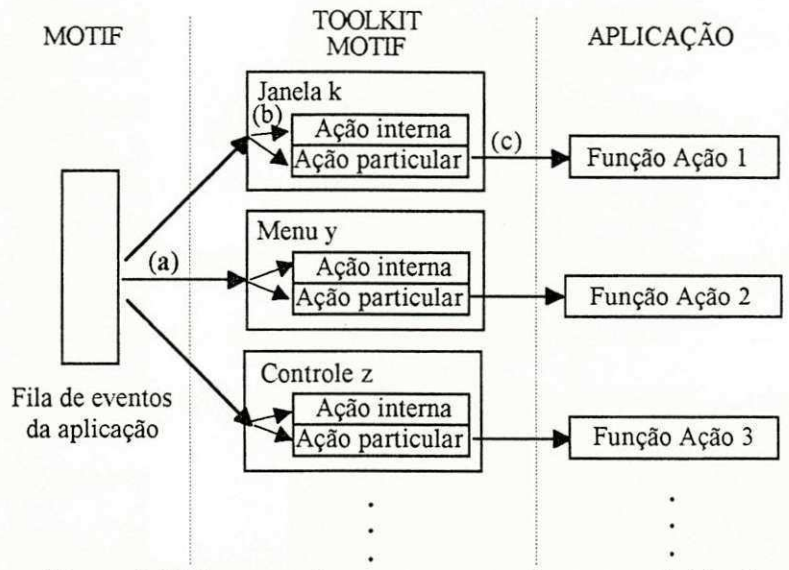

Figura 3.12 Esquema de processamento de eventos do Motif

3.3.4. Modelo de Saída Gráfica

Esta seção compara o modelo de saída gráfica do Windows, com o modelo de saída gráfica do Motif. Inicialmente serão discutidas as características gerais das bibliotecas de programação disponíveis nos dois ambientes e em seguida será estudado o modo de definição dos atributos gráficos adotado por cada GUI.

3.3.4.1. Biblioteca de Saída Gráfica

O Windows oferece ao programador uma biblioteca de saída gráfica independente de dispositivo. Esta biblioteca, denominada *Graphic Device Interface* (GDI), oferece funções para a definição dos atributos das operações gráficas, criação de texto, geração de primitivas gráficas e apresentação de *bitmaps*. O sistema de coordenadas usado pela biblioteca GDI pode ser tanto um sistema de coordenadas virtual, quanto o sistema de coordenadas baseado em *pixels*. Quando o

programador utiliza o sistema de coordenadas virtual, as unidades virtuais serão automaticamente convertidas para *pixels* pela GDI. O modo de desenho adotado pela biblioteca GDI é imediato. Isto significa que, depois que uma imagem gráfica for mostrada na tela, nenhuma cópia desta imagem será mantida pelo sistema na memória do computador. Desta forma, é de total responsabilidade da aplicação redesenhar as imagens geradas por ela, sempre que for necessário.

No Motif, a biblioteca disponível para a criação de saídas gráficas é a Xlib. A Xlib oferece funções para a definição dos atributos gráficos, criação de primitivas gráficas, apresentação de texto e *bitmaps*. O sistema de coordenadas oferecido pela Xlib é baseado apenas em *pixels* e o seu modo de desenho também é imediato.

3.3.4.2. Contexto do Dispositivo Versus Contexto Gráfico

O **contexto do dispositivo** é o conjunto de atributos definido pelas aplicações Windows para a realização das operações de saída gráfica. O contexto do dispositivo contém informações a respeito de um dispositivo de saída gráfica em particular, uma permissão de acesso e um conjunto de atributos gráficos associados a este dispositivo. As informações sobre o dispositivo de saída gráfica têm a finalidade de permitir que as aplicações localizem fisicamente o dispositivo que desejam acessar; a permissão de acesso permite que o Windows coordene o acesso das várias aplicações aos dispositivos compartilhados; os atributos gráficos definem as características visuais das operações realizadas no dispositivo. Os atributos gráficos do Windows agrupam-se formando os seguintes objetos lógicos: lápis, pincel, fontes e cor.

- **Lápis**: é o objeto lógico usado para desenhar linhas. É composto pelo conjunto de atributos que especifica a cor, a largura e o tipo de uma linha.

- **Pincel**: é usado para preencher áreas. Os atributos que compõem um pincel são o estilo, a cor e o padrão de enchimento.

• **Fontes**: são objetos lógicos requeridos para se escrever texto. Os fontes são padrões que descrevem a forma e o tamanho de cada letra, número e sinais de pontuação.

• **Cor**: é um atributo usado para definir a cor dos demais objetos lógicos.

O **contexto gráfico** do Motif é equivalente ao contexto do dispositivo no Windows. É um conjunto de atributos, mantido no servidor X, que é utilizado durante as operações gráficas das aplicações Motif. A manutenção destes atributos no servidor X ajuda a melhorar o desempenho das GUIs desenvolvidas para ambientes de redes, pois reduz o tráfego através da rede dos parâmetros necessários à realização das operações gráficas. O contexto gráfico define atributos como largura da linha, estilo, cor, padrão de enchimento de áreas e fontes, por exemplo. Cada atributo no Motif, no entanto, é definido de uma forma individual, não existe o conceito de objeto lógico.

De todos os atributos que são necessários para a realização das operações gráficas, apenas os mais complexos serão apresentados detalhadamente neste trabalho: fontes e cores. Uma rápida descrição de como estes atributos são definidos no Windows e no Motif será apresentada a seguir.

Fontes

O Windows coloca à disposição do programador vários tipos tipos de fontes: *bit-mapped*, escaláveis, Adobe e TrueType. A seguir serão discutidas as características de cada um desses fontes[17].

As primeiras versões do Windows colocaram à disposição do programador fontes do tipo *bit-mapped*. Apesar desse tipo de fonte oferecer ao usuário e ao programador fontes de diferentes estilos e tamanhos, a apresentação de um texto com fontes *bit-mapped* no vídeo raramente

[17] É importante salientar que alguns tipos de fontes, como o Adobe e o TrueType, apenas passaram a estar disponíveis em versões mais recentes do Windows (a partir da versão 3.0).

corresponde à saída exata gerada pela impressora. Isso ocorre porque a única forma de garantir essa correspondência seria dispor de um *bitmap* particular para cada estilo e tamanho de fonte desejado, desenvolvido especificamente para cada dispositivo de *hardware* existente no mercado. Essa forte dependência de *hardware,* característica dos fontes *bit-mapped,* tornam esse tipo de fonte difícil de manipular. Já para se obter fontes *bit-mapped* de vários tamanhos, normalmente duplica-se o número de linhas e colunas do *bitmap* que representa o fonte. Esse método, entretanto, pode gerar distorções nos caracteres do fonte.

A solução posteriomente adotada para resolver o problema de escalabilidade dos fontes *bit-mapped* no Windows foram os fontes escaláveis. Esse tipo de fonte baseia-se em pontos fixos, pré-definidos para cada caractere, conectados por segmentos de retas. Devido à sua filosofia de criação, os fontes escaláveis podem ser facilmente comprimidos ou expandidos. Contudo, esse método pode ser interessante para dispositivos como os *plotters*[18], mas é tipograficamente complexo para outros tipos de dispositivos.

As distorções apresentadas pelos fontes devido à sua expansão só passaram a ser corrigidas no Windows após o lançamento do utilitário Adobe Type Manager na sua versão para Windows. Esse utilitário, desenvolvido pela Adobe Systems Inc., permite que aplicações Windows utilizem fontes compatíveis com PostScript[19], colocando à disposição do programador fontes escaláveis continuamente de altíssima qualidade e oferecendo equivalência entre os fontes utilizados pelos vídeos e pelas impressoras.

Com o lançamento do Windows 3.1, uma nova tecnologia passou a estar disponível no Windows, o TrueType[20]. O TrueType é um padrão aberto[21] para fontes que pode ser utilizado por

---

[18] Os *plotters* são dispositivos de saída gráfica utilizados para desenhar mapas, diagramas e outras imagens gráficas baseadas em linhas.

[19] O PostScript é uma linguagem de alto nível, independente de dispositivos, desenvolvida pela Adobe Systems Inc. para a especificação de serviços gráficos. Essa linguagem permite a produção de imagens de altíssima qualidade, compativelmente, em qualquer impressora ou dispositivo de vídeo que disponha de um interpretador PostScript.

[20] O TrueType é uma tecnologia de fontes, originalmente desenvolvida pela Apple, que promete oferecer WYSIWYG (What

qualquer dispositivo de saída gráfica oferecido pelo Windows. Os fontes TrueType oferecidos pelo Windows baseiam-se em um princípio semelhante ao utilizado pelos fontes PostScript, estando disponíveis em todos os dispositivos de saída, todas as resoluções de vídeo e em todos os tamanhos de fontes.

As funções, disponíveis na API Windows, para a manipulação de fontes permitem: definir as características lógicas do fonte a ser utilizado, selecionar um fonte físico a partir de um fonte lógico, obter as características de um determinado fonte físico e liberar um fonte lógico. Além dos fontes pré-definidos, o Windows também permite que o programador crie fontes em estilos e tamanhos próprios.

Os fontes disponíveis no Motif são fontes *bit-mapped*. Existem fontes pré-definidos em vários estilos e tamanhos. Como já discutido anteriormente, esse tipo de fonte está disponível apenas em tamanhos fixos, apresenta uma forte dependência de *hardware* e por isso não é portável. A falta de um padrão aberto para os fontes no Motif dificulta bastante a manipulação dos fontes neste ambiente. Para se especificar o tipo do fonte a ser utilizado em uma operação de escrita no Motif, o nome do fonte deve ser fornecido como parâmetro durante a criação da cadeia.

Cores

Tanto o Windows quanto o Motif baseiam-se no sistema RGB (Red, Green, Blue) para a geração de cores. Nesse sistema, as cores são definidas através da especificação da intensidade de três cores primárias: vermelho, verde e azul. Como um *byte* é reservado para a especificação de cada uma dessas cores, esse sistema oferece cerca de 16 milhões de combinações únicas para

---

You See Is What You Get - as informações apresentadas ao usuário na tela possuem forma, tamanho e estilo correspondentes às imagens obtidas através da impressora, durante a impressão das informações) para os ambientes Macintosh e Windows. Esta tecnologia é um padrão aberto que tende a se tornar o padrão *de facto* para fontes no mercado.

[21] Um padrão aberto é um padrão que pode ser adotado por qualquer fabricante. Isso significa que qualquer fabricante pode criar fontes no formato TrueType.

cores. Por limitações de *hardware*, no entanto, esse potencial, normalmente, não é totalmente utilizado.

No Windows, o programador dispõe de três métodos para a definição de cores: pelo valor absoluto da cor, pelo índice da paleta de cores ou pela especificação da mistura de cores.

Para definir uma cor através do **valor absoluto da cor**, o programador especificará a intensidade das cores vermelha, verde e azul, em uma escala numérica de 0 a 255, para obter a cor desejada. As variações de cores obtidas através desse método vão depender do tipo de dispositivo utilizado [Norton90].

Para definir uma cor através do **índice da paleta de cores,** o programador especificará apenas o índice que corresponde à cor desejada em uma tabela de triplas RGB, pré-definida pelo Windows. Esse método permite que um programa defina as cores a serem representadas nos *slots* disponíveis no *hardware*.

Já para definir uma cor através da **especificação da mistura de cores,** o programador especificará a intensidade das cores vermelha, verde e azul, para obter a cor desejada, de forma semelhante à definição de cores pelo valor absoluto da cor. Apesar da aparente semelhança com o método do valor absoluto da cor, o método da especificação da mistura de cores permite ao programador misturar cores puras e dar origem a novas cores. Maiores informações sobre o sistema de definição de cores no Windows pode ser encontrado na referência [Norton90].

De forma semelhante ao Windows, o Motif permite a especificação de cores puras através do seu valor absoluto ou através do índice de um mapa de cores. O mapa de cores é uma tabela semelhante à paleta de cores do Windows. O Motif, no entanto, permite que a aplicação defina o seu próprio mapa de cores, o que não é permitido no Windows. Uma outra facilidade oferecida pelo Motif é um banco de cores que permite ao programador acessar as cores através do seu

nome. Outras informações sobre o sistema de definição de cores no Motif podem ser encontradas na referência [Nye90].

3.3.5. Modelo dos Objetos da Interface

O terceiro modelo a ser estudado é o modelo de programação dos objetos da GUI. Nesta seção serão apresentadas as características dos objetos disponíveis nas duas GUIs e o modelo de programação de cada objeto no Windows será comparado ao respectivo modelo de programação no Motif.

3.3.5.1. Tipos de Objetos

Os objetos disponíveis nas GUIs Windows e Motif podem ser classificados de forma geral em: janelas, menus, quadros de diálogo e ícones. A seguir serão apresentados os tipos de janelas, menus e quadros de diálogo que podem ser encontrados nos dois ambientes gráficos.

Janelas

Existem três tipos básicos de janelas que podem ser identificadas em uma aplicação Windows: a janela principal, a janela secundária e a janela de controle.

A **janela principal** é a primeira janela a ser mostrada por uma aplicação que está sendo executada. Esta janela apresenta para o usuário os principais comandos e opções disponíveis na aplicação e serve como janela de referência básica.

As **janelas secundárias**, por sua vez, são usadas em complemento às janelas principais, quando é necessário mostrar ou obter informações adicionais, em resposta às ações do usuário. Este tipo de janela pode ter como pai uma janela principal ou outra janela secundária. São as janelas secundárias, por exemplo, que são usadas para se criar os quadros de diálogo[22].

---

[22] Devido a suas características particulares, os quadros de diálogo serão discutidos separadamente.

Já as **janelas de controle** são janelas especiais do Windows, através das quais os controles de uma aplicação podem ser criados. Este tipo de janela é oferecido pelo Windows para permitir que uma janela principal ou secundária seja subdividida em áreas funcionais menores e assim possa ser mais facilmente gerenciada. Devido a esse fato, uma janela de controle somente poderá ser criada como filha de uma janela principal ou secundária.

As janelas encontradas na GUI Motif são de dois tipos: janela principal e janela temporária. A **janela principal** do Motif tem a mesma funcionalidade da janela principal do Windows e a **janela temporária** é equivalente à janela secundária do Windows.

No Motif, não existe uma janela específica que seja equivalente à janela de controle do Windows. Esse tipo de facilidade não é necessária no Motif devido à independência funcional inerente a cada objeto da GUI (vide seção 3.3.3.3). Como os objetos da GUI Motif já são independentemente gerenciáveis, não há necessidade do programador subdividir as janelas existentes em áreas funcionais menores para facilitar o seu gerenciamento.

Menus

Três tipos básicos de menus podem ser encontrados em uma aplicação Windows: menu *dropdown*, menu contextual e menu em cascata.

O **menu dropdown** é representado por um título e é usado em conjunto com outros menus *dropdowns*, para compor a barra de menu de uma janela (figura 3.13). Para que as opções do menu *dropdown* sejam mostradas, o usuário deve ativá-lo. Este menu é comumente encontrado nas aplicações.

O **menu contextual** é um menu flutuante que só é mostrado quando chamado explicitamente pelo usuário. O tipo de comando associado aos itens de um menu contextual depende da localização do apontador quando o botão do *mouse* for pressionado e, por isto, estes

menus oferecem uma forma rápida de acessar comandos.

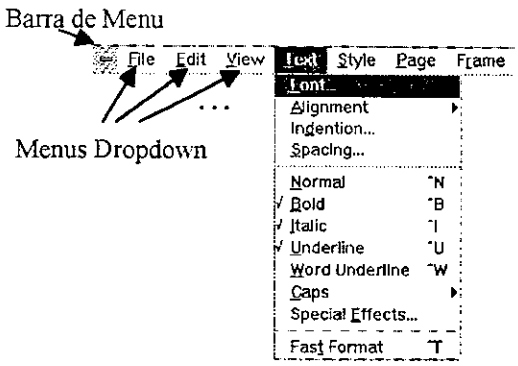

Figura 3.13 Barra de menus composta por menus dropdown.

Por exemplo, se o botão do *mouse* for pressionado quando o apontador estiver localizado na área de desenho de um editor gráfico, os itens do menu contextual ativado dirão respeito a comandos relacionados à criação de formas geométricas (Figura 3.14).

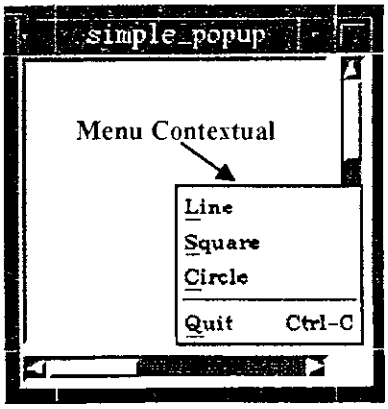

Figura 3.14 Menu contextual de um editor gráfico simples.

O menu em cascata é um submenu mostrado a partir da ativação de um item de menu. Esse menu sempre aparece à direira do item selecionado e pode ser ativado a partir do item de um menu *dropdown*, de um menu contextual ou mesmo de um outro menu em cascata (figura 3.15).

Os tipos de menus disponíveis em uma aplicação Motif são: menu *dropdown*, menu contextual, menu em cascata e menu de opções. Os três primeiros menus possuem aparência e funcionalidade semelhantes aos menus disponíveis no Windows.

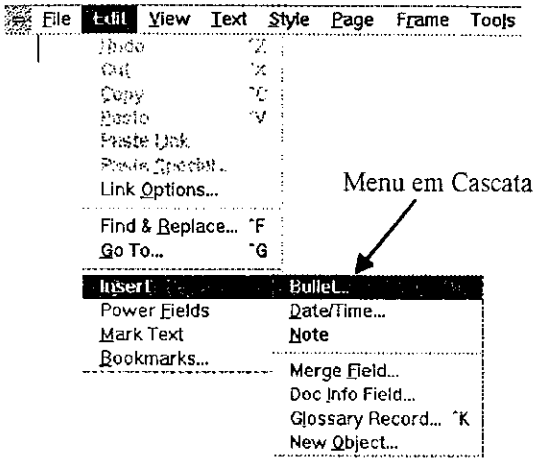

Figura 3.15 Exemplo de menu em cascata ativado a partir de um menu dropdown.

O menu de opções, no entanto, é um tipo de menu disponível apenas no Motif. Esse tipo de menu é uma combinação entre um quadro de lista e um menu *dropdown* (Figura 3.16). A opção apresentada pelo menu é uma das alternativas da lista de itens. Quando o menu é ativado, todas as opções disponíveis na lista serão visíveis ao usuário. Isto lhe permite escolher uma outra opção.

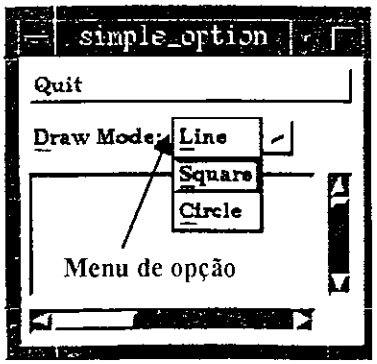

Figura 3.16 Menu de opção do Motif.

Quadros de Diálogo

Os quadros de diálogo de uma GUI podem ser classificados de acordo com a influência que exercem sobre as aplicações, quando estão mapeados na tela. De acordo com esse critério, tanto o Windows quanto o Motif oferecem ao programador os mesmos tipos de quadros de

diálogo. Nessas duas GUIs os quadros de diálogo podem ser classificados em: com modo na aplicação, com modo no sistema e sem modo.

• **Quadro de diálogo com modo na aplicação**: é um tipo de quadro de diálogo, que quando ativado, desabilita a interação do usuário com as demais janelas da aplicação. O usuário pode, no entanto, interagir com qualquer outra aplicação do sistema. Este tipo de quadro de diálogo deve ser usado quando as informações requeridas pelo quadro de diálogo tiverem que ser fornecidas, para que a ação, requerida pelo usuário, possa ser executada. O quadro de diálogo apresentado ao usuário, após a ativação do comando de abertura de arquivos em um editor de texto, é um exemplo de um quadro de diálogo com modo na aplicação.

• **Quadro de diálogo com modo no sistema**: é um tipo de quadro de diálogo que, quando apresentado na tela, impossibilita o usuário de realizar qualquer operação, em qualquer aplicação do sistema. Deve-se usar este tipo de quadro de diálogo quando a informação a ser obtida ou fornecida pelo quadro de diálogo for importante para todas as aplicações em execução no sistema. Um exemplo da utilização deste tipo de quadro de diálogo é o aviso da ocorrência de erros irrecuperáveis no sistema.

• **Quadro de diálogo sem modo**: permite que o usuário realize qualquer operação, em qualquer aplicação do sistema, mesmo quando o quadro de diálogo estiver mapeado na tela. Este tipo de quadro de diálogo deve ser usado quando as informações ou opções do quadro de diálogo não requiserem atenção imediata da aplicação.

Controles

Todos os tipos de controles apresentados na seção 2.3.3.3 estão disponíveis tanto no Windows quanto no Motif. Controles do tipo botões, quadros de tique, quadros de lista, texto, quadros de incremento e suas subdivisões internas podem ser encontrados como componentes das

janelas nos dois ambientes gráficos. Pode-se dizer que para todo controle disponível no Windows, existe um controle correspondente no Motif, com aparência e funcionalidade bastante semelhantes.

Os quadros de lista, no entanto, podem ser de dois tipos no Windows e de apenas um no Motif. No Windows, os quadros de lista podem ser simples ou *combo box*. A lista simples é semelhante ao quadro de lista mostrado na seção 2.3.3.3. É este tipo de lista que também está disponível no Motif. O *combo box*, no entanto, é uma variação que só está disponível no Windows. Este tipo de quadro de lista é uma combinação entre um controle do tipo texto com um controle do tipo quadro de lista simples. A figura 3.17 apresenta um quadro de diálogo com três *combo box*: um para a especificação do fonte de texto; outro para a especificação do atributo do texto e outro para a especificação do tamanho do fonte. O controle do tipo texto do *combo box* pode ser um texto estático ou um quadro de texto editável, e a lista de opções pode ser aberta (os itens da lista são permanentemente mostrados), ou fechada (apenas o item selecionado é permanentemente mostrado).

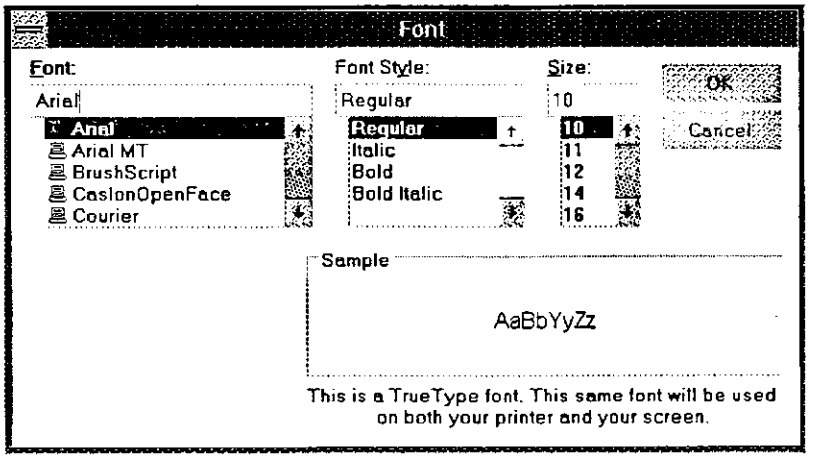

Figura 3.17 Quadro de diálogo com três combo boxes.

Ícones

No Windows um ícone é um tipo especial de *bitmap*. Um *bitmap* é uma imagem gráfica representada por uma matriz de *bits*, armazenadas na memória do computador, onde um ou mais *bits* correspondem a um *pixel* no vídeo.

De forma semelhante aos ícones no Windows, os ícones no Motif também são representados por *bitmaps*.

3.3.5.2. Criação dos Objetos

A fim de comparar o modelo de programação dos objetos da GUI nos ambientes Windows e Motif, os passos necessários para a criação de janelas, menus, quadros de diálogo e ícones, nas duas GUI, serão apresentados a seguir.

Janelas

No Windows, toda janela pertence a uma determinada classe. A classe da janela define o estilo, a função-ação, o ícone, o cursor e a barra de menu da janela. O Windows exige que a classe associada à janela seja sempre registrada antes da criação da janela. Após o registro da classe, as funções de criação e apresentação da janela, disponíveis na API Windows, podem ser utilizadas (Figura 3.18a).

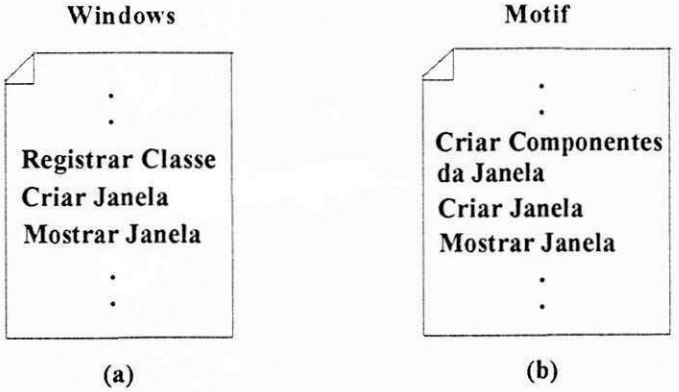

Figura 3.18 Criação de janelas no Windows (a) X criação de janelas no Motif (b).

Já no Motif, não apenas para a criação de janelas, mas para a criação de qualquer objeto da GUI é necessário que se especifique uma classe. Entretanto, o conjunto de classes pré-definidas pela API Motif, na maioria dos casos, é suficiente para que as aplicações sejam desenvolvidas sem que seja necessário o registro de novas classes.

Para se criar uma janela no Motif o usuário deve proceder da seguinte forma: criar, com antecedência, todos os componenetes da janela (como por exemplo, barra de menu e barras de rolamento); criar a janela desejada especificando, como parâmetros, os componentes criados anteriormente e mostrar a janela criada. Todas estas operações podem ser realizadas utilizando funções de conveniência da API Motif (Figura 3.18.b).

Menus

Dois métodos estão disponíveis para se criar um menu no Windows. O primeiro é definir o menu no arquivo de recursos. O segundo é criar o menu através das funções de suporte. Estes dois métodos serão discutidos a seguir.

• **Definir o menu no arquivo de recursos:** esta é a forma mais simples de se criar um menu no Windows. O arquivo de recursos é um arquivo que contém definições de objetos da GUI que podem ser armazenados como recursos. Os recursos são dados *read-only* que são carregados pela aplicação na medida em que forem necessários. Como o menu é um dado *read-only*, ele pode ser definido pelo programador no arquivo de recursos. Para isto o programador utiliza o comando MENU, especificamente definido no Windows para este propósito. A figura 3.19 mostra a definição de um menu através deste comando.

```
1 MENU
{
        POPUP "&File"
        {
                MENUITEM    "&New"          , 1
                MENUITEM    "&Open..."      , 2
                MENUITEM    "&Save"         , 3
        }
        POPUP "&Edit"
        {
                MENUITEM    "&Undo"         , 4
                MENUITEM    SEPARADOR
                MENUITEM    "Cu&t"          , 5
                MENUITEM    "&Paste"        , 6

        }
}
```

Figura 3.19 Exemplo de definição de menu no arquivo de recursos.

Antes da palavra chave MENU, o identificador associado ao menu deve ser especificado. No exemplo acima, o identificador do menu é "1". Em seguida os submenus ou itens, que compõem o menu, são definidos. Nesse exemplo dois submenus foram definidos através do comando POPUP, os títulos destes submenus são "&File" e "&Edit". O "&" no título especifica a *hotkey*[23] associada ao submenu. Dentro de cada comando POPUP, especificam-se os comandos do tipo MENUITEM, para a definição dos itens do submenu, ou mesmo outros comandos POPUP. No exemplo acima apenas itens do tipo ação e separadores foram definidos. Maiores detalhes sobre a utilização do arquivo de recurso no Windows podem ser encontrados em [Petzold90] e [Norton90].

• **Criar o menu através das funções de suporte**: a segunda alternativa disponível é definir todo o menu em tempo de execução. Neste caso, dentro do próprio código da aplicação, a função de criação de menus é utilizada para criar um menu vazio. Em seguida, para cada item que

[23] Neste trabalho o termo *hotkey* refere-se a um mnemônico, normalmente associado a um item de menu, que pode ser ativado através do teclado. A *hotkey* é uma facilidade que vai permitir ao programador selecionar um item de menu sem que seja necessário fazer uso do apontador.

se deseja acrescentar ao menu, faz-se uma chamada individual à função que cria um item de menu. Depois que todos os itens forem especificados, o menu criado poderá ser autônomo ou definido como submenu de algum outro menu.

No Motif, a única forma de se criar um menu é através da utilização das funções de suporte da API Motif. De forma semelhante ao segundo método de criação de menus do Windows, os menus são definidos em tempo de execução. A API Motif oferece uma função específica para a criação de cada tipo de menu disponível na GUI. Os parâmetros exigidos por cada função, para a criação de um menu são: pai, função-ação do menu e lista de itens. Um menu do Motif também pode ser definido como submenu de um outro menu.

Quadros de Diálogo

O Windows oferece duas formas para se criar um quadro de diálogo. A primeira é definir o quadro de diálogo no arquivo de recursos. A segunda é criar o quadro de diálogo através das funções de suporte.

• **Definição do quadro de diálogo no arquivo de recursos**: de forma semelhante a um menu, um quadro de diálogo também pode ser armazenado como recurso. Desta forma, o programador pode definir o quadro de diálogo no arquivo de recursos da aplicação, através do comando DIALOG. A figura 3.20 mostra como o comando DIALOG pode ser usado para definir um quadro de diálogo.

```
IDD_ABOUT DIALOG LOADONCALL MOVEABLE DISCARTABLE          20, 24, 180, 84
STYLE   WS_DLGFRAME | WS_POPUP
{
   CONTROL "Exemplo", IDD_TEXT1, "static", SS_CENTER | WS_CHILD, 47, 13, 84, 8
   CONTROL "de quadro de diálogo", IDD_TEXT2, "static", SS_CENTER | WS_CHILD, 0, 30, 180, 8
   CONTROL "Ok", IDD_OK, "button", BS_DEFPUSHBUTTON | WS_CHILD 70, 64, 40, 16
}
```

Figura 3.20 Exemplo de definição de um quadro de diálogo no arquivo de recursos.

O primeiro parâmetro do comando DIALOG define o identificador do quadro de diálogo. No exemplo, a constante IDD_ABOUT representa esse identificador. Depois da palavra chave DIALOG são especificadas as opções de memória do quadro de diálogo. Em seguida a localização e o tamanho da janela são definidos. No exemplo acima, a coordenada superior do canto esquerdo da janela é definida como 20, 24 e a coordenada do canto inferior direito é 180, 84. A unidade utilizada na definição das coordenadas é o *pixel*.

A segunda linha de comando define o tipo da janela a ser criada para o quadro de diálogo. No exemplo acima é definida uma janela secundária (WS_POPUP) com borda fixa (WS_DLGFRAME).

Logo após o comando STYLE seguem os controles do quadro de diálogo, definidos através do comando CONTROL. No exemplo apresentado foram definidos dois controles do tipo texto estático e um controle do tipo botão de comando. O primeiro parâmetro do comando CONTROL é o título do controle. Em seguida o identificador de cada controle é definido. No exemplo acima esses identificadores são IDD_TEX1 para o primeiro controle; IDD_TEXT2 para o segundo e IDD_OK para o terceiro. Na seqüência, a classe e os atributos de cada controle são definidos. Os atributos SS_CENTER | WS_CHILD definem os dois primeiros textos estáticos como controles centralizados. Já os atributos BS_DEFPUSHBUTTON | WS_CHILD definem o último controle como botão de comando *default* do quadro de diálogo. Maiores informações sobre o comando DIALOG podem ser encontradas em [Petzold90] e [Norton90].

• **Criar o quadro de diálogo através das funções de suporte**: os quadros de diálogo podem ser criados através das funções de suporte para a criação de janelas, disponíveis na API Windows. O programador pode criar uma janela secundária e nela incluir os controles desejados. A mesma função de criação de janelas é usada, tanto para a criação da janela secundária quanto

para a criação dos controles das janelas.

O Motif também coloca à disposição do programador duas formas de criar um quadro de diálogo: usar funções pré-definidas ou criar novas variedades de quadros de diálogo. Esses dois métodos serão apresentados sucintamente a seguir.

• **Usar Funções Pré-definidas**: esta é a forma mais fácil de se criar um quadro de diálogo no Motif. A API Motif oferece ao programador um conjunto de funções pré-definidas que lhe permite criar, diretamente, alguns tipos de quadros de diálogo. Os quadros de diálogo que podem ser criados desta forma são: quadros de mensagem e quadros de seleção. Um quadro de mensagem apresenta ao usuário uma simples mensagem; e um quadro de seleção é um quadro de diálogo que possui um quadro de lista.

• **Criar Novos Quadros de Diálogo**: quando for necessário criar um quadro de diálogo, diferente dos oferecidos pelas funções pré-definidas, o programador deverá combinar as várias classes de objetos disponíveis na API Motif, de forma a obter o quadro de diálogo desejado. Essa tarefa, no entanto, pode se tornar bastante complexa. Detalhes sobre este assunto podem ser encontrados na referência [Heller91].

Icones

Para criar um ícone no Windows, o programador define o gráfico que o representará, através de ferramentas como o SDKPAINT (*Windows Software Development Kit*) ou o *Resource Workshop* (compilador Borland C++). Em seguida, como um ícone no Windows também pode ser armazenado como recurso, um editor de arquivos é usado para incluir no arquivo de recursos o comando de definição do ícone da aplicação.

Para criar um ícone no Motif, o programador pode usar um editor de *bitmap*, chamado Bitmap, que, normalmente, está disponível no ambiente Motif. Uma vez na aplicação, o *bitmap* é

carregado através de uma função específica e é associado à janela principal da aplicação.

Até este capítulo foram especificados os requisitos básicos e específicos da biblioteca UNIFIC e apresentadas as características mais importantes das GUIs a serem unificadas. Com base nesses dados torna-se possível definir o modelo de programação da biblioteca UNIFIC. Esse é o assunto a ser discutido no próximo capítulo.

Capítulo 4

4. Modelo de Programação da UNIFIC

Este capítulo define o modelo de programação adotado pela biblioteca UNIFIC. A definição deste modelo baseia-se nos requisitos básicos e específicos da biblioteca UNIFIC, definidos no capítulo 2, e no estudo dos modelos de programação das GUIs Windows e Motif, apresentado no capítulo 3. Os seguintes aspectos funcionais do modelo de programação UNIFIC serão descritos: serviços do sistema operacional, troca de dados entre as aplicações, entrada de dados orientada a eventos, saída gráfica e objetos da GUI. A medida em que o modelo de programação UNIFIC for sendo descrito, será apresentada a lista de funções da API UNIFIC correspondente à funcionalidade discutida. Informações detalhadas sobre cada uma dessas funções poderão ser encontradas no apêndice A.

4.1. Serviços do Sistema Operacional

A biblioteca UNIFIC oferece ao programador várias funções de acesso aos serviços do sistema operacional. Estas funções divididem-se em quatro grupos: gerenciamento de memória, entrada e saída em arquivos, execução de aplicações e manipulação de cadeias. Várias outras classes de funções poderiam ter sido oferecidas, entretanto, o conjunto de funções selecionadas, por se tratar de serviços comumente encontrados em qualquer sistema operacional, é o suficiente para atender às necessidades mais comuns da maioria das aplicações.

4.1.1. Gerenciamento de Memória

Toda área de memória alocada dinamicamente por uma aplicação UNIFIC é do tipo global e fixa (as características deste tipo de memória foram discutidas na seção 3.3.1.2). Para permitir a gerencia desse tipo de memória, a API UNIFIC coloca à disposição do programador funções de alocação, relocação e liberação de áreas memória. Depois de alocar uma área de memória, o programador pode usar diretamente o endereço de memória retornado pela função de alocação para acessar a área alocada. Quando uma área alocada não for mais útil para a aplicação que a alocou, esta área deverá ser liberada. A Tabela 4.1 apresenta uma rápida descrição das funções disponíveis para o gerenciamento de memória na API UNIFIC.

Função	Descrição
U_AlocaMemoria	Aloca uma área na memória.
U_LiberaMemoria	Descarta uma área de memória.
U_RealocaMemoria	Realoca uma área ne memória.

Tabela 4.1 Lista de funções UNIFIC para gerenciamento de memória.

A implementação desse esquema de gerência de memória em alguns ambientes gráficos, como o Windows por exemplo, pode impor ao programa algum *overhead.*

Áreas de memória alocadas dinâmicamente, através da API UNIFIC, só podem ser compartilhadas entre aplicações por meio do mecanismo de *clipboard* (vide seção 4.2.1).

4.1.2. Entrada e Saída em Arquivo

A biblioteca UNIFIC oferece um conjunto de funções que permitem ao programador realizar operações de entrada e saída de dados em arquivos. As funções disponíveis na UNIFIC permitem ao programador abrir, posicionar, ler, escrever e fechar um arquivo.

Para que seja possível realizar operações sobre um arquivo, a UNIFIC exige que ele esteja aberto. Durante a abertura de um arquivo o programador deve especificar o nome, o modo de abertura e o modo de acesso do arquivo que deseja acessar. O **modo de abertura** especifica o tipo de operação que a aplicação poderá realizar no arquivo aberto. Na UNIFIC, os modos de abertura disponíveis são: escrita, leitura, leitura e escrita de dados. Se um arquivo aberto para escrita não existir, este arquivo será criado. O **modo de acesso** do arquivo define se o arquivo será acessado com exclusividade ou não. A exclusividade de acesso só deve ser requerida quando o programador precisar garantir que só ele tem acesso a um determinado arquivo, durante a realização de operações de escrita ou acréscimo de dados. Depois que um arquivo for aberto, operações como a leitura, escrita ou posicionamento (*seek*) dentro de um arquivo podem ser efetuadas. Quando todas as operações desejadas tiverem sido realizadas, o arquivo deverá ser fechado. A Tabela 4.2 apresenta uma descrição sucinta das funções disponíveis para a geração de entrada e saída em arquivos na API UNIFIC.

Funções	Descrição
U_AbreArquivo	Abre um arquivo.
U_EscreveArquivo	Escreve dados em um arquivo.
U_FechaArquivo	Fecha um arquivo.
U_LeArquivo	Lê dados de um arquivo.
U_PosicionaArquivo	Posiciona o apontador de leitura ou escrita em um arquivo.

Tabela 4.2 Lista de funções da API UNIFIC para entrada e saída em arquivos.

4.1.3. Execução de Aplicações

A biblioteca UNIFIC oferece uma função própria que permite a um processo pai ativar a execução de um outro (processo filho). Essa função retorna para o processo pai o *status* final de

execução do processo filho. Se ocorrer algum erro durante a execução do processo filho, ou se por algum motivo o processo não puder ser criado, um código de erro apropriado é retornado para o processo pai. A tabela 4.3 apresenta a única função disponível na API UNIFIC para a execução de aplicações UNIFIC.

Função	Descrição
U_ExecAplic	Permite que uma aplicação UNIFIC ative a excução de uma outra aplicação.

Tabela 4.3 Função UNIFIC para execução de aplicações.

Esse tipo de função é válido, do ponto de vista do programador, pois a detecção dos erros ocorridos durante a execução do processo filho torna-se transparente. A tarefa do programador ficará resumida à definição do tratamento a ser dados aos erros que, por ventura, venham a ocorrer.

4.1.4. Manipulação de Cadeias

As cadeias, em uma aplicação UNIFIC, são definidas como uma sequência de caracteres definida no próprio código fonte da aplicação. A manipulação de cadeias é feita através de um conjunto de funções específicas, de forma semelhante à manipulação de cadeias na linguagem "C". A UNIFIC oferece ao programador funções para a concatenação, comparação, cópia e obtenção do tamanho das cadeias. As funções disponíveis na API UNIFIC para a manipulação de cadeias serão apresentadas na tabela 4.4.

Funções	Descrição
U_ComparaCadeia	Compara duas cadeias de caracteres.
U_ConcatenaCadeia	Concatena duas cadeias de caracteres.
U_CopiaCadeia	Copia o conteúdo de uma cadeia de caracteres para outra.
U_ObtemTamanhoCadeia	Determina o tamanho de uma cadeia de caracteres.

Tabela 4.4 Lista de funções da API UNIFIC para a manipulação de cadeias.

O esquema de manipulação de cadeias adotado pela UNIFIC não permite que as cadeias de caracteres sejam definidas no arquivo de recursos da aplicação, nem oferece, a princípio, facilidades para a especificação da direção da cadeia ou do conjunto de caracteres a ser utilizado. Para aumentar a potencialidade das operações oferecidas pela UNIFIC para a manipulação de cadeias, um estudo mais detalhado sobre este assunto deverá ser realizado no futuro.

4.2. Troca de Dados entre Aplicações

O mecanismo oferecido pela biblioteca UNIFIC para a troca de dados entre aplicações é o *clipboard*. Esse mecanismo de troca de dados está disponível na UNIFIC por ser um esquema de troca de dados eficiente, podendo ser implementado em qualquer ambiente que ofereça um serviço de memória compartilhada. As funções disponíveis para a utilização desse mecanismo pelas aplicações UNIFIC e uma idéia geral de como utilizá-las, serão apresentadas a seguir.

4.2.1. Uso do Clipboard

A UNIFIC oferece funções que permitem ao programador escrever e ler dados no *clipboard*, além de funções para consultar o formato ou remover os dados contidos no *clipboard*. Durante as operações de escrita ou leitura, é necessário que o programador especifique o formato dos dados manipulados. Os formatos de dados previstos pela biblioteca UNIFIC são: caractere e *bitmap*.

Antes que uma aplicação coloque os dados desejados no *clipboard*, os dados nele existentes devem ser removidos. Em seguida, o programador pode utilizar a função de escrita de dados no *clipboard* para armazenar os dados a serem compartilhados. A aplicação que deseja obter os dados, por sua vez, pode consultar o formato dos dados armazenados no *clipboard* e

verificar se são realmente de seu interesse. Caso afirmativo, a aplicaçao lê os dados contidos no *clipboard* e faz uma cópia desses dados para a sua aplicação. A lista das funções disponíveis na API UNIFIC para a troca de dados através do *clipboard* será apresentada na tabela 4.5.

Funções	Descrição
U_AbreClipboard	Abre o *clipboard*.
U_ApagaDadosClipboard	Remove os dados contidos no *clipboard*.
U_EscreveDadosClipboard	Copia dados para o *clipboard*.
U_FechaClipboard	Fecha o *clipboard*.
U_LeDadosClipboard	Obtem os dados contidos no *clipboard*.
U_VerificaFormatoClipboard	Verifica se um determinado formato de dados está contido no *clipboard*.

Tabela 4.5 Lista de funções da API UNIFIC para troca de dados através do clipboard.

O fato de apenas o mecanismo de *clipboard* estar à disposição do programador UNIFIC para a troca de dados entre aplicações, limita, em muitos aspectos, o tipo de informação que pode ser trocada entre as aplicações UNIFIC (vide seção 3.3.2.2). Logo que for possível implementar um método semelhante ao da troca dinâmica de dados do Windows, de forma eficiente, na maioria dos ambientes gráficos, esse tipo de troca de dados também será oferecido pela API UNIFIC.

4.3. Entrada de Dados Orientada a Eventos

Como visto na seção 3.3.3, o sistema de entrada de dados orientado a eventos de uma GUI pode ser dividido em três fases: geração do evento, recebimento do evento e processamento do evento. Dessas três fases, apenas a primeira continua a ser realizada pela GUI do ambiente em que a UNIFIC está sendo executada, as fases de recebimento e processamento de eventos passam a ser de responsabilidade da própria UNIFIC.

4.3.1. Geração dos Eventos

Cada GUI continua a aceitar as mesmas fontes de geração de eventos (*hardware*, GUI ou aplicação), sem interferências da UNIFIC. Os eventos gerados seguem o curso já decrito na seção 2.3.3.1 até chegarem à fila de eventos da aplicação destino. A partir desse ponto, os eventos passam a ser gerenciados pela biblioteca UNIFIC através de um esquema particular de recebimento e processamento de eventos.

Os eventos que se originam do *hardware* ou da GUI são gerados de forma automática do ponto de vista do programador. Entretanto, se o programador desejar que sua aplicação envie algum evento para uma determinada janela, ele deverá fazer uso de uma função específica para este fim. A tabela 4.6 apresenta uma descrição sucinta dessa função.

Função	Descrição
U_EnviaEventJan	Envia um determinado evento para uma janela específica.

Tabela 4.6 Função da API UNIFIC para a geração de eventos pela aplicação.

4.3.2. Recebimento dos Eventos

Para que uma aplicação UNIFIC obtenha os eventos da sua fila de eventos, o programador deve usar uma função especial que implementa um laço de recebimento de eventos para as aplicações UNIFIC. O laço de eventos, além de ler os eventos da fila de eventos da aplicação, interpreta-os e os envia para a função de processamento adequada. Os eventos recebidos por uma aplicação UNIFIC, durante a sua execução, classificam-se basicamente em: internos e externos.

O **eventos internos** são importantes apenas para o controle interno da UNIFIC. Por isso esses eventos são sempre processados de forma *default*, sem que seja permitido ao programador influenciar nas ações associadas a este tipo de evento.

Já os **eventos externos** são aqueles eventos que servem de elo de ligação entre a interação do usuário com os objetos da GUI e as aplicações. São os eventos externos aos quais o programador pode associar uma ação dentro da função-ação. A UNIFIC oferece onze eventos[24] externos que são o suficiente para o programador definir a funcionalidade associada a maioria das aplicações. Alguns destes eventos indicam a criação, restauração ou o fechamento de uma janela. Outros, indicam os movimentos do apontador, a ocorrência de caracteres provenientes do teclado e a ativação de itens de menus ou controles de quadros de diálogo. Futuramente, outros eventos poderão ser acrescidos ao conjunto de eventos externos da UNIFIC. Por exemplo, eventos específicos para troca de dados através do *clipboard* e eventos para definição de operações de rolamento de tela.

Apesar da biblioteca UNIFIC oferecer um laço padrão, também é possível para o programador definir o seu próprio laço de recebimento de eventos. Isso pode ser necessário se o programador precisar de uma funcionalidade que não seja prevista pelo laço padrão. Para construir seu próprio laço, o programador pode utilizar qualquer função disponível na API UNIFIC, mas deve incluir, necessariamente, as funções que, separadamente, lêem os eventos da fila (removendo-os ou não), identificam-nos e os enviam para processamento. A tabela 4.7 apresenta as funções disponíveis na API UNIFIC para a criação de um laço de evento particular.

Funções	Descrição
U_ConsultaFilaEvento	Obtem o próximo evento, da fila de eventos da aplicação, sem removê-lo da fila.
U_EnviaEventProc	Envia um evento para ser processado pela função-ação adequada.
U_LacoPrincipal	Lê, interpreta e envia para processamento os eventos contidos na fila de eventos da aplicação.

[24] Uma lista de todos os eventos externos da biblioteca UNIFIC pode ser encontrada no apêndice A1.4.

Funções	Descrição
U_ObtemEvento	Lê o próximo evento da fila de eventos da aplicação.

Tabela 4.7 Funções disponíveis na API UNIFIC para a criação de um laço de recebimento de eventos.

É importante observar que uma aplicação UNIFIC deve ter sempre um laço destinado ao processamento dos eventos enviados à aplicação. Do contrário torna-se impossível a interação entre o usuário e a aplicação.

4.3.3. Processamento dos Eventos

Na UNIFIC, as funções de processamento de eventos, ou funções-ação, cujo conceito foi definido na seção 3.3.3.3, estão associadas às janelas da aplicação. Cada função-ação de uma aplicação UNIFIC deve, portanto, conter as ações associadas aos eventos de interesse da janela e a todos os demais objetos que compõem esta janela. Por exemplo, se a janela possuir uma barra de menu, a ação a ser realizada quando um evento indicar a ativação de um dos itens do menu deve estar especificada na função-ação da janela. Se a janela for um quadro de diálogo, por outro lado, a função-ação desta janela deve definir as ações a serem realizadas quando seus controles forem ativados.

O fato da função-ação ser associada ao objeto janela na biblioteca UNIFIC oferece ao programador as seguintes vantagens: consistência, reaproveitamento das características da janela e facilidade de manutenção.

- **Consistência**: todo comportamento associado a uma determinada janela, inclusive o comportamento associado aos seus objetos componentes, será encontrado em uma única função.

- **Reaproveitamento das características da janela**: será fácil para o programador criar mais de uma janela com o mesmo comportamento, basta apenas que a mesma função-ação seja associada a mais de uma janela.

• **Facilidade de manutenção**: será fácil para o programador localizar e modificar qualquer ação associada a uma determinada janela.

Como as funções-ação de uma aplicação devem ser definidas, integralmente, pelo próprio programador, a única função pré-definida pela UNIFIC que se relaciona ao processamento dos eventos é a função-ação *default* da API UNIFIC. Essa função é apresentada na tabela 4.8.

Função	Descrição
U_FuncAcDef	Processa os eventos UNIFIC de forma *default*.

Tabela 4.8 Função dispovível na API UNIFIC para processamento de eventos.

4.4. Geração da Saída Gráfica

A UNIFIC coloca à disposição do programador um conjunto de funções destinada à geração de saída gráfica. Dentre as funções oferecidas, as seguintes classes de funções se destacam: definição dos atributos gráficos, geração de formas geométricas e apresentação de texto. A filosofia de programação adotada por cada uma dessas classes será apresentada a seguir:

4.4.1. Definição dos Atributos Gráficos

Entre todas as funções oferecidas pela biblioteca UNIFIC para a geração de operações gráficas, o único conjunto que adota uma filosofia particular de programação, em relação aos modelos de programação estudados neste trabalho, é o de definição dos atributos gráficos. Na biblioteca UNIFIC os objetos lógicos são definidos em relação à uma janela. Quando o programador desejar realizar uma operação gráfica e não quiser que os atributos *defaults* do ambiente sejam utilizados, ele deverá criar uma **ferramenta de desenho**. A ferramenta de desenho é uma ferramenta lógica que associa a uma janela os seguintes objetos lógicos: lápis, pincel, fonte e cor. Desta forma, quando o usuário realizar uma operação de saída gráfica em uma

determinada janela, os objetos lógicos associados a esta janela através da ferramenta de desenho serão utilizados.

A tabela 4.9 oferece uma descrição rápida das funções disponíveis na UNIFIC para a manipulação dos atributos gráficos.

Função	Descrição
U_DefineAtribGrafic	Define o conjunto de atributos gráficos a ser utilizado por uma janela.
U_ObtemAtribGrafic	Obtem o conjunto de atributos gráficos definidos para uma janela.

Tabela 4.9 Funções UNIFIC para a gerência dos atributos gráficos.

Algumas considerações a respeito dos fontes e cores disponíveis na biblioteca UNIFIC serão apresentadas a seguir.

4.4.1.1. Fontes

A biblioteca UNIFIC oferece ao programador fontes do tipo *bit-mapped*. Para utilizar um determinado fonte, a aplicação deverá definir uma ferramenta de desenho que especifique as características do fonte desejado. Três fontes básicos pré-definidos são oferecidos pela UNIFIC : System, Courier e Helvética. O fonte System coloca à disposição do programador o fonte *default* do dispositivo que está sendo utilizado pela aplicação. O fonte Courier oferece ao programador um fonte com largura constante para todos os seus caracteres. Finalmente, o fonte Helvética oferece ao programador um conjunto de caracteres cuja largura varia de acordo com as características físicas de cada caractere representado. Se o fonte selecionado pelo programador não estiver disponível no ambiente gráfico onde a aplicação UNIFIC está sendo executada, ele será substituído por outro fonte correspondente do ambiente.

Se, durante uma operação de escrita, o fonte a ser utilizado não for especificado pelo

programador, a UNIFIC adotará sempre o fonte *default* do dispositivo de saída gráfico utilizado.

Do ponto de vista do programador, o fato da API UNIFIC oferecer apenas fontes *bit-mapped* é uma desvantagem, devido às limitações inerentes a este tipo de fonte. No futuro, entretanto, será desenvolvido um estudo específico para definir um conjunto de fontes compatíveis com PostScript para as aplicações UNIFIC.

4.4.1.2. Cores

As cores no UNIFIC baseiam-se no sistema RGB (vide seção 3.3.4.2), por ser este o sistema de cores adotado pela maioria dos vídeos de hoje [Nye90]. A definição de cores no UNIFIC é feita através da função **U_DefAtribGrafic** disponível na API UNIFIC que permite a especificação do valor absoluto da cor, de forma semelhante ao explicado na seção 3.3.4.2.

4.4.2. Geração de Formas Geométricas

A UNIFIC oferece ao programador um conjunto de funções básicas para a criação de formas geométricas. Inicialmente, as formas geométricas serão sempre apresentadas na área interna de uma determinada janela. A tabela 4.10 descreve sucintamente as funções da biblioteca UNIFIC para a criação de formas geométricas.

Funções	Descrição
U_Arco	Desenha um arco elíptico.
U_Elipse	Desenha uma elipse.
U_Linha	Desenha uma linha.
U_Polígono	Desenha um polígono com dois ou mais vétices.
U_Ponto	Desenha um ponto.
U_Retangulo	Desenha um retângulo.

Tabela 4.10 Funções disponíveis na API UNIFIC para criação de formas geométricas.

4.4.3. Apresentação de Texto

Os textos são tratados pela API UNIFIC como objetos gráficos. Assim sendo, durante sua apresentação é necessário que o programador especifique sua localização, em *pixel*, em relação à área interna da janela. A tabela 4.11 apresenta as funções disponíveis na API UNIFIC para a apresentação de texto.

Função	Descrição
U_EscreveTexto	Escreve um texto com linha única.
U_EscreveTextoFormatado	Escreve um texto com várias linhas.

Tabela 4.11 Funções disponíveis na API UNIFIC para a apresentação de texto.

4.5. Gerência dos Objetos da Interface do Usuário

Nesta seção será apresentado o modelo de programação dos objetos da GUI. Inicialmente serão especificados os tipos de objetos da GUI oferecidos pela biblioteca UNIFIC e, em seguida, serão definidos os modelos de programação adotados pela UNIFIC para a criação de cada um destes objetos.

4.5.1. Tipos de Objetos da Interface do Usuário

A biblioteca UNIFIC oferece ao programador um conjunto básico de objetos da GUI. De uma forma geral os objetos podem ser classificados em: janelas, menus, controles e ícones.

4.5.1.1. Janelas

Três tipos básicos de janelas podem ser criados através da biblioteca UNIFIC: janela principal, janela secundária e janela de controle. A janela principal deve ser criada como a primeira janela de uma aplicação. Enquanto que a janela secundária deve ser usada quando o

programador precisar de uma janela que não seja a janela principal. A janela de controle, por sua vez, é uma janela especial que deve ser utilizada apenas quando o programador precisar definir controles em uma janela principal. Isto será necessário em determinadas classes de aplicações, como o desenvolvimento de uma calculadora, por exemplo.

A UNIFIC permite que os seguintes componentes sejam definidos para uma janela principal ou secundária: ícones de maximização e minimização, menu do sistema[25], barra de menu, título, borda redimensionável ou fixa, e barras de rolamento. As diversas combinações possíveis entre os tipos de janelas e seus componentes, permitem ao programador UNIFIC criar diversos tipos de janelas. No entanto, algumas restrições devem ser observadas: a borda redimensionável e a barra de menu, não devem ser utilizadas em janelas de quadro de diálogo; as barras de rolamento não devem ser usadas na borda de um quadro de diálogo.

4.5.1.2. Menus

A biblioteca UNIFIC oferece ao programador dois tipos básicos de menus: *dropdown* e cascata. Os menus *dropdown* podem ser usados pelo programador para definir os itens da barra de menu de uma janela e os menus em cascata podem ser usados para definir os submenus associados a cada item de um menu *dropdown*. Os itens que compõem um menu UNIFIC podem ser do tipo comando ou opções, de forma semelhante aos itens definidos na seção 2.3.3.3.

Por uma questão de simplificação, os menus contextuais, normalmente disponíveis na maioria das GUI, não serão oferecidos, a princípio, pela API UNIFIC. Esta simplificação implica que o usuário deve sempre navegar por barras de menus para ativar os comandos da aplicação. Apesar dessa limitação não ser, na prática, um transtorno para o usuário, no futuro esse tipo de

---

[25] O menu do sistema é um menu padrão, que pode estar presente na borda de qualquer janela, cujos comandos estão relacionados à gerência da janela. Minimização, maximização, movimento ou mudança de tamanho de uma janela são alguns dos comandos que podem ser encontrados nesse tipo de menu.

menu será acrescentado ao conjunto de objetos da API UNIFIC.

4.5.1.3. Controles

Os controles disponíveis para uma aplicação UNIFIC são: botões de comandos, botões de rádio, quadros de marcar, texto estático, quadros de texto editável e quadros de lista simples. Esses são os controles básicos oferecidos pela biblioteca UNIFIC, mas o programador poderá combinar alguns destes controles entre si para dar origem a novos controles. Por exemplo, um quadro de lista do tipo *combo box* pode ser criado por uma aplicação UNIFIC através da combinanção de um quadro de lista simples com um quadro de texto editável. Note-se no entanto que um *combo box* no Motif vai ser representado por uma lista simples, uma vez que este tipo de controle não está disponível no ambiente Motif. Já um quadro de incremento pode ser criado combinando-se um quadro de texto editável com dois ícones de comando.

Alguns outros controles, comuns a várias GUIs, que não foram colocados à disposição do programador na primeira versão da UNIFIC, serão oferecidos em versões futuras. A barra de deslizamento é um dos exemplos desse tipo de controle.

4.5.1.4. Ícones

A biblioteca UNIFIC permite aos programadores de aplicação utilizarem os ícones para representar uma aplicação cuja janela principal foi minimizada. Inicialmente, apenas por simplificação da API UNIFIC, o uso de ícones como comando ou dados não estará disponível. Na prática isto significa que aplicações, como por exemplo editores gráficos, não poderão oferecer ao usuário de barras de comandos que utilizem símbolos gráficos. Em um estudo futuro, essa utilização para os ícones também será oferecida pela API UNIFIC.

4.5.2. Criação dos Objetos

O modelo de criação dos objetos da GUI pode ser melhor estudado quando dividido em criação de janelas, criação de menus, criação de quadros de diálogo e criação de ícones (Vide seção 3.3.5.2).

4.5.2.1. Janelas

A biblioteca UNIFIC coloca à disposição do programador uma função que permite a criação de diversos tipos de janela. O tipo da janela criada pela função dependerá apenas dos valores fornecidos como parâmetros à função. Os parâmetros requeridos pela função de criação de janelas especificam o tipo básico da janela (principal, secundária ou de controle), seu estilo (pai, localização, função-ação) e seus componentes (tipo da borda, barra de menu, barras de rolamento). Para a criação de uma janela principal, por exemplo, normalmente devem ser definidos, no mínimo, uma barra de título, uma barra de menu, um menu do sistema, ícones de comando para maximização e minimização da janela e o ícone da aplicação. Já uma janela secundária, criada para compor um quadro de diálogo, por exemplo, geralmente não possui barra de menu.

Para facilitar a tarefa de criação de janelas, a biblioteca UNIFIC oferece vários *flags* de estilo pré-definidos para a criação de diversos tipos de janelas. Se, no entanto, a janela que o programador deseja criar não se enquadrar em nenhum dos estilos pré-definidos, o programador poderá, então, combinar os estilos básicos oferecidos pela UNIFIC para criar a janela desejada. Maiores informações a respeito da função de criação de janelas e seus *flags* de estilo podem ser encontradas no apêndice A.

A tabela 4.12 apresentará a lista das funções disponíveis na API UNIFIC para a criação e gerência das janelas.

Funções	Descrição
U_AtribuiJanFoco	Define a janela que deve receber a entrada proveniente do *mouse* ou *teclado*.
U_CapturaTeclado	Faz com que toda entrada proveniente do *mouse* passe a ser enviada para uma determinada janela.
U_CriaJanela	Cria uma janela..
U_DestroiJanela	Destroi uma janela.
U_FechaJanela	Fecha (transforma em ícone) uma janela.
U_LiberaCaptura	Libera a captura do *mouse* e volta a processar a entrada de forma convencional.
U_LocalizaJanela	Obtém a posição e o tamanho de uma janela.
U_MostraJanela	Torna uma janela visível para o usuário.
U_MoveJanela	Muda a posição e o tamanho de uma janela.
U_ObtemEstadoJanela	Obtem o estado atual da janela.
U_ObtemJanCaptura	Obtem a janela que está capturando a entrada proveniente do *mouse*.
U_ObtemJanelaAtiva	Obtem o identificador da janela ativa.
U_ObtemJanFoco	Obtem o identificador da janela que posui o foco de entrada.
U_ObtemPaiJanela	Obtem o identificador da janela pai de uma determinada janela.

Tabela 4.12 Lista das funções disponíveis na API UNIFIC para criação e gerência de janelas.

4.5.2.2. Menus

Os menus de uma aplicação UNIFIC são sempre criados a partir do seu arquivo de recursos (Vide seção 3.3.5.2). O programador deve utilizar o comando MENU[26], no arquivo de recursos, para definir as características do menu desejado. Para carregar esse menu durante a execução da aplicação, o programador utiliza uma função específica oferecida pela API UNIFIC para este fim. Se, no entanto, o menu definido for utilizado como barra de menu para uma janela, não é necessário que a aplicação carregue especificamente este menu. Quando a janela for

[26] Detalhes sobre o formato do comando MENU podem ser encontrados no apêndice B.

apresentada ao usuário, o menu será automaticamente criado.

O método de definição de menus no arquivo de recursos foi adotado pela biblioteca UNIFIC por oferecer as seguintes facilidades: facilidade de manutenção e facilidade para o desenvolvimento de ferramentas de criação.

- **Facilidade de Manutenção**: o código das aplicações ficará mais leve e, consequentemente, mais fácil de manter, pois as definições de todos os menus da aplicação estarão contidas no arquivo de recursos. Uma outra facilidade de manutenção é que apenas o arquivo de recursos precisará ser recompilado em caso de mudança nas características de algum dos menus da aplicação.

- **Facilidade para o Desenvolvimento de Ferramentas de Criação**: o fato dos menus serem especificados através de um formato específico em um arquivo separado do resto do fonte da aplicação, permitirá o desenvolvimento de ferramentas gráficas que facilitarão a criação dos menus.

A tabela 4.13 apresenta as funções disponíveis na API UNIFIC para a criação e gerência dos menus da aplicação.

Funções	Descrição
U_AtivaItem	Torna disponível ou indisponível para seleção pelo usuário um item do menu.
U_CarregaMenu	Cria um menu a partir de informações do arquivo de recursos.
U_MarcaItem	Marca ou desmarca um item de menu do tipo opção.
U_ObtemEstadoItem	Obtem o estado de um item de menu.
U_ObtemMenu	Obtem o identificador da barra de menu associada a uma determinada janela.

Tabela 4.13 Lista de funções disponíveis na UNIFIC para a criação e gerência dos menus.

4.5.2.3. Quadros de Diálogo

Os quadros de diálogo são janelas que apresentam características muito específicas e por isto sua criação é tratada em separado.

Os quadros de diálogo nas aplicações UNIFIC também são criados a partir do arquivo de recursos. O programador deve utilizar o comando DIALOGO[27], no arquivo de recursos, para definir as características do quadro de diálogo desejado.

Três tipos de quadros de diálogo podem ser criados por uma aplicação UNIFIC: com modo na aplicação, sem modo na aplicação e com modo no sistema. Depois que o quadro de diálogo for definido no arquivo de recursos, ele será carregado para a aplicação através da função de criação de quadros de diálogo oferecida pela API UNIFIC. As vantagens do método de criação dos quadros de diálogo no UNIFIC são as mesmas da criação de menus, já discutidas na seção anterior.

Existe, no entanto, uma outra forma de se criar um quadro de diálogo na UNIFIC. Para permitir que controles possam ser usados como componentes de uma janela principal, a UNIFIC permite que eles sejam criados como filhos de uma janela principal a partir da função de criação de janelas. Para tornar a criação de janelas ortogonal portanto, a UNIFIC permite que os controles também possam ser associados a janelas secundárias, permitindo assim a criação de quadros de diálogo a partir da função de criação de janelas. Apesar deste método estar disponível para a criação dos quadros de diálogo, não é aconselhável usá-lo, pois as vantagens proporcionadas pelo arquivo de recursos da aplicação não seriam obtidas.

A tabela 4.14 apresentará uma lista das funções disponíveis no UNIFIC para a criação e gerência dos controles UNIFIC.

---

[27] Informações a respeito do comando DIALOGO podem ser encontradas no apêndice B.

Funções	Descrição
U_AtribuiIntDlgInt	Atribui um inteiro à área editável de um quadro de texto editável.
U_AtribuiCadeiaDlgItem	Atribui uma cadeia de caracteres à área editável de um quadro de texto editável.
U_CriaDialogo	Cria um quadro de diálogo a partir do arquivo de recursos.
U_CriaDlgMsg	Cria um quadro de mensagem.
U_DestroiDialogo	Destroi um quadro de diálogo.
U_EstaMarcadoRM	Indica se um controle do tipo botão de rádio ou quadro de marcar está marcado.
U_MarcaBotãoMarc	Marca ou desmarca um controle do tipo quadro de marcar.
U_MarcaBotaoRadio	Marca um controle do tipo botão de rádio.
U_MostraLista	Inicializa um controle do tipo quadro de lista com sua lista de opções.
U_MostraListaDir	Inicializa um controle do tipo quadro de lista com os arquivos contidos em um determinado diretório.
U_ObtemEstadoControle	Obtem o estado de uma janela do tipo controle.
U_ObtemIdControle	Obtem o identificador da janela do tipo controle.
U_ObtemIntDlgItem	Obtem o inteiro associado à área editável de um quadro de texto editável.
U_ObtemOpcaoLista	Obtem a opção selecionada pelo usuário em um quadro de lista.
U_ObtemCadeiaDlgItem	Obtem a cadeia associada à área editável de um quadro de texto editável.

Tabela 4.14 Lista de funções para a criação e gerência de controles na UNIFIC.

4.5.2.4. Ícones

Os ícones das aplicações UNIFIC podem ser criados através de uma ferramenta específica: o editor de recursos. O editor de recursos é uma ferramenta, oferecida pela própria biblioteca UNIFIC, para a criação de qualquer recurso da aplicação. No caso da criação de ícones, essa ferramenta criará um arquivo *bitmap* que será compatível em qualquer ambiente gráfico no qual a aplicação venha a ser executada. A definição do nome através do qual a aplicação acessará o

ícone deverá estar presente no arquivo de recursos.

Como os ícones na API UNIFIC só poderão ser utilizados como representação de uma janela, não é necessário que a API UNIFIC ofereça nenhuma função específica para sua manipulação. Quando utilizado para representar janelas, o ícone é carregado automaticamente no ato de criação da janela.

Uma vez que o modelo de programação da biblioteca UNIFIC foi definido, é necessário descrever, na prática, como este modelo de programação pode ser utilizado pelo programador para desenvolver aplicações UNIFIC. Este é o assunto a ser discutido no próximo capítulo.

Capítulo 5

5. Programando com a Biblioteca UNIFIC

Este capítulo mostra ao programador como desenvolver uma aplicação UNIFIC. Essa apresentação será dividida em duas partes. A primeira parte discutirá os fundamentos da programação UNIFIC. Para atingir este objetivo, será analisada a filosofia de programação UNIFIC, serão apresentadas as ferramentas necessárias para o seu desenvolvimento e sugeridos alguns passos para o desenvolvimento dessas aplicações. Na segunda parte serão discutidos os aspectos práticos da programação UNIFIC. Para atingir a este fim, será feita a análise do código de três exemplos UNIFIC. O primeiro exemplo mostrará como criar janelas; o segundo como definir um menu e associá-lo a uma janela e o terceiro como criar quadros de diálogo.

5.1. Filosofia de Programação UNIFIC

Antes de começar a projetar uma aplicação UNIFIC, o programador deve familiarizar-se com a filosofia de programação UNIFIC. O primeiro passo, nesse sentido, é conhecer as fases típicas de uma aplicação UNIFIC. Isto é importante, pois oferece subsídios para que o programador estruture corretamente a aplicação que está sendo desenvolvida.

A estrutura geral de uma aplicação UNIFIC apresenta três fases bem definidas: fase de inicialização, fase de criação e fase de execução (figura 5.1). As operações normalmente realizadas em cada uma dessas fases serão discutidas a seguir.

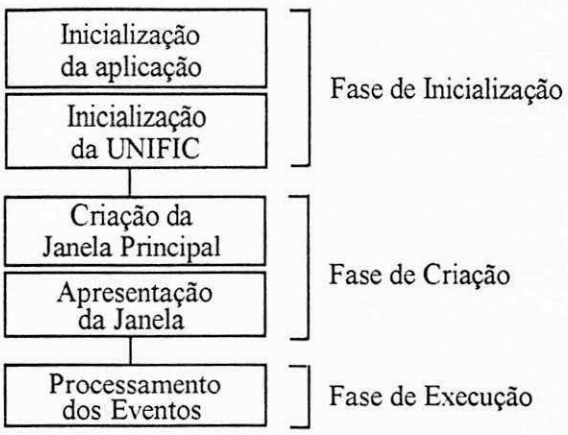

Figura 5.1 Principais fases de uma aplicação UNIFIC

5.1.1. Fase de Inicialização

Na fase de inicialização são realizados dois tipos de operação: inicialização da aplicação e inicialização da UNIFIC.

- **Inicialização da Aplicação**: é caracterizada pela realização das operações de inicialização, particulares a cada aplicação.

- **Inicialização da UNIFIC**: é caracterizada pela chamada à função **U_InicializaAplicacao()**. Antes que qualquer outra função da biblioteca UNIFIC possa ser utilizada, a função **U_InicializaAplicacao()** deve ser chamada. Esta função realiza as operações de inicialização necessárias para a execução das aplicações em qualquer ambiente gráfico. É esta função que, dentre outras tarefas, estabelece contato com a GUI do ambiente destino e lê as informações do arquivo de recursos[28].

5.1.2. Fase de Criação

Dois tipos de operação caracterizam a fase de criação de uma aplicação UNIFIC: criação da janela principal e apresentação da janela para o usuário.

---

[28] Os detalhes a respeito do arquivo de recursos da UNIFIC foram explicados na seção 4.5.2.2 e 4.5.2.3.

• **Criação da Janela Principal**: toda aplicação UNIFIC deve possuir pelo menos uma janela: a janela principal de uma aplicação. A criação dessa janela caracteriza a primeira parte da fase de criação da aplicação UNIFIC. As janelas da aplicação são criadas através da função **CriaJanela()**.

• **Apresentação da Janela**: as janelas criadas não são automaticamente visíveis para o usuário logo após a sua criação. Para torná-las visíveis é necessário que elas sejam explicitamente mapeadas na tela. Esta tarefa é realizada através da função **U_MostraJanela()**. A chamada a esta função ainda caracteriza a fase de criação de uma aplicação UNIFIC.

5.1.3. Fase de Execução

A última fase de uma aplicação UNIFIC é a fase de execução. Esta fase é caracterizada pelas operações de processamento dos eventos.

• **Processamento dos Eventos**: a melhor forma de se processar um evento é através de um laço infinito que espera a ocorrência dos eventos destinados à aplicação. Este laço deve ler os eventos da fila de eventos da aplicação, interpretá-los e enviá-los para a função-ação adequada. A UNIFIC oferece ao programador um laço pré-definido, a função **U_LacoPrincipal()**, que realiza todas essas operações. A chamada à função **U_LacoPrincipal()** caracteriza a fase de execução das aplicações UNIFIC.

A fase de execução também inclui as definições das funções-ação, associadas às janelas da aplicação, ou qualquer outra função necessária ao funcionamento da aplicação.

5.2. Ferramentas Necessárias

Para desenvolver a maioria das aplicações UNIFIC o programador deverá dispor, em seu ambiente de trabalho, do seguinte conjunto de ferramentas: editor de texto, compilador "C" ou "C++" e editor de recursos.

• **Editor de Texto**: será utilizado para criar os arquivos fontes (.C ou .CPP, e .H) da aplicação.

• **Compilador C ou C++**: será necessário para compilar os fontes em "C" ou "C++" da aplicação.

• **Editor de Recursos**: é um editor especial, que será necessário para a criação dos recursos utilizados pela aplicação (ícones, menus e quadros de diálogo).

O editor de texto e o compilador de linguagem "C" ou "C++" podem ser escolhidos a critério do programador. O editor de recursos, no entanto, é uma ferramenta fornecida junto com a própria biblioteca UNIFIC.

5.3. Passos para Criar uma Aplicação UNIFIC

Os seguintes passos devem ser seguidos, para o desenvolvimento de uma aplicação UNIFIC (figura 5.2):

1. Criar os arquivos fontes da aplicação (a aplicação deve conter pelo menos a função *main()* e as funções-ação associadas às janelas da aplicação).

2. Usar o editor de recursos para criar os recursos da aplicação (ícones, menus, quadros de diálogo).

3. Definir os recursos criados no passo 2, através de comandos apropriados, no arquivo de recursos (.REC) (durante a execução da aplicação, o arquivo executável consultará as

informações do arquivo de recursos).

4. Compilar e ligar todos os fontes escritos em linguagem "C" ou "C++".

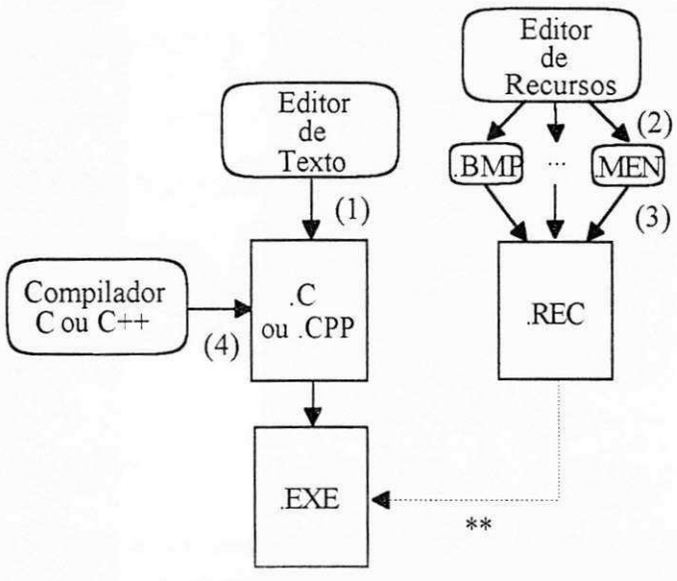

Figura 5.2 Passos para o desenvolvimento de uma aplicação UNIFIC.

5.4. Exemplos Práticos de Aplicações UNIFIC

Esta seção mostrará como os fundamentos da programação UNIFIC, apresentados nas seções anteriores, poderão ser aplicados na prática, através da análise de três exemplos. Os exemplos foram elaborados de forma a abranger os aspectos mais importantes da programação UNIFIC. O primeiro exemplo mostra como criar a janela principal de uma aplicação, o segundo mostra como criar a barra de menu de uma janela e o terceiro mostra como criar quadros de diálogo. Durante a discussão de cada um dos exemplos serão explicados os detalhes do seu código fonte e apresentada a saída gerada pela aplicação, quando executada no ambiente Windows. Nos exemplos apresentados, o arquivo de recursos de cada aplicação foi criado manualmente através de um editor de texto convencional. No futuro, quando o editor de recursos UNIFIC estiver disponível, este artifício não será necessário.

Embora nos exemplos apresentados, a maioria das funções utilizadas pertençam à biblioteca UNIFIC, qualquer função da biblioteca "C" poderia ter sido utilizada. Recomenda-se ainda que o programador recorra ao apêndice A para esclarecer qualquer dúvida a respeito da interface ou da funcionalidade de qualquer função UNIFIC.

Como aspectos de internacionalização não foram especificamente tratados neste trabalho, durante a apresentação dos exemplo assume-se que o conjunto de caracteres utilizados para a definição das cadeias de caracteres é o ISO8859/1[29].

5.4.1. Criando Janelas

O primeiro exemplo a ser discutido mostra como criar a janela principal de uma aplicação. Esta janela suporta as seguintes operações: maximização (janela passa a ocupar toda a tela), minimização (janela transforma-se em ícone), mudança de localização ou mudança de tamanho. A figura 5.3 mostra o resultado da execução desta aplicação no ambiente Windows.

Figura 5.3 Resultado da execução da Basico.exe em um ambiente Windows.

O código fonte desta aplicação é apresentado na figura 5.4.

```
/************************************************

Basico.C

************************************************/
#include    "Unific.h"

#define     nomeRec     "basico.rec"
```

[29] ISO8859/1 é um conjunto de caracteres padrão de oito *bits*, chamado Latin1, estabelecido pela International Standards Organization (ISO). Esse conjunto de caracteres acomoda caracteres de pontuação, dígitos, letras, letras acentuadas e outros símbolos especiais necessários às linguagens européias ocidentais.

```
int   main();
long  FuncJanPrinc(  U_IDJAN, U_EVENTO );

main()
{
      U_INDJAN    indJan;          /* identificador da janela principal */
      U_EVENTO    apEvento;        /* apontador para estrutura de eventos */

      U_InicializaAplicacao( nomeRec );
      indJan = U_CriaJanela( NULO,        /* identificador do pai da janela */
                  J_JANPRINC | JE_VISIVEL,/* estilo da janela */
                  "Basico",               /* título da janela */
                  CJ_DEFAULT,             /* posição horizontal default */
                  CJ_DEFAULT,             /* posição vertical default */
                  CJ_DEFAULT,             /* largura default */
                  CJ_DEFAULT,             /* altura default */
                  NULL,                   /* nome do menu da janela */
                  "IconeBasico",          /* nome no ícone da janela */
                  FuncJanPrinc            /* função da janela */
                  );
      if( indJan == NULO ){
            return FALSE;                 /* janela não pôde ser criada */
      }

      U_MostraJanela( idJan );

      U_LacoPrincipal( apEvento );


      return TRUE;
}

long FuncJanPrinc( idJan, apEvento )
U_IDJAN     idJan;
U_EVENTO    apEvento;
{

      switch( U_TIPO( apEvento ) ){
            case E_DESTROI:
                  U_DestroiJanela( idJan );
                  break;
            default:
                  return( U_FuncAcDef( apEvento ) );
      }
      return NULL;
}

/*************************************************

Basico.REC

*************************************************/
ICONE       iconeBasico IcoBasic.bmp
```

Figura 5.4 Aplicação Basico.c

5.4.1.1. Procedimento Principal

A primeira função definida no código fonte da aplicação apresentada é a função **main()**. Esta função determina o ponto de entrada da aplicação. No corpo da função **main()** o seguinte trecho de código é encontrado:

```
U_InicializaAplicacao( nomeRec );
```

A função **U_InicializaAplicacao()** realiza todas as operações de inicialização necessárias à biblioteca UNIFIC. Esta função recebe como parâmetro uma cadeia de caracteres contendo o nome do arquivo de recursos da aplicação. Esta é a única função que compõe a fase de inicialização da aplicação exemplo.

5.4.1.2. Criação das Janelas

Na fase de criação da aplicação Basico.c, apenas a janela principal é criada. O trecho de código que corresponde à criação dessa janela é apresentado a seguir:

```
indJan = U_CriaJanela(  NULO,        /* identificador do pai da janela */
           J_JANPRINC | JE_VISIVEL,/* estilo da janela */
           "Basico",                 /* título da janela */
           CJ_DEFAULT,               /* x - posição horizontal default */
           CJ_DEFAULT,               /* y - posição vertical default  */
           CJ_DEFAULT,               /* cx - largura default */
           CJ_DEFAULT,               /* cy - altura default */
           NULL,                     /* nome do menu da janela */
           "IconeBasico",            /* nome no ícone da janela */
           FuncJanPrinc              /* função da janela */
           );
```

A função **U_CriaJanela()** é a função oferecida pela biblioteca UNIFIC para a criação de qualquer tipo de janela. Esta função exige que o programador defina dez parâmetros durante sua utilização. Os valores fornecidos à função como parâmetros permitem a criação dos mais diversos tipos de janelas. No exemplo discutido, os valores especificados permitem a criação de uma janela

do tipo principal:

O primeiro parâmetro da função **U_CriaJanela()** especifica o identificador do pai da janela criada. Como no exemplo apresentado a janela criada não tem pai, este parâmetro foi definido como NULO[30].

O segundo parâmetro da função **U_CriaJanela()** define o estilo da janela a ser criada. A biblioteca UNIFIC oferece um conjunto de *flags* pré-definidos para a especificação do estilo da janela. No exemplo acima os *flags* utilizados foram J_JANPRINC e JE_VISIVEL. O *flag* J_JANPRINC é utilizado para indicar que a janela criada deve ser do tipo janela principal. O *flag* JE_VISIVEL indica que a janela deverá ser visível para o usuário. Uma lista de todos os *flags* que podem ser utilizados como estilo, durante a criação de uma janela, pode ser encontrada na seção A1.6.1 do apêndice A.

O terceito parâmetro indica o título da borda da janela. Nesse exemplo o título da janela é a palavra "Básico".

O quarto, quinto, sexto e sétimo parâmetros são usados para indicar a localização e o tamanho da janela. Os parâmetros quarto e o quinto indicam a coordenada superior do canto esquerdo da janela. Os parâmetros sexto e o sétimo indicam a coordenada inferior do canto direito da janela. A unidade utilizada na definição esses parâmetros é o *pixel*. No exemplo apresentado, a definição dos parâmetros de localização é feita da forma mais simples possível: utilizando-se uma constante pré-definida pela UNIFIC para este propósito, a constante CJ_DEFAULT. Esta constante oferece um valor *default* para as coordenadas da janela.

O oitavo parâmetro indica o nome da barra de menu da aplicação. No exemplo acima o valor fornecido para este parâmetro é NULL[31]. Isto significa que a janela criada não possui barra

---

30 NULO é uma constante, definida pela biblioteca UNIFIC, que pode ser utilizada pelo programador para indicar que uma variável inteira refere-se a um valor nulo.

31 NULL é uma constante, definida pela linguagem "C", para indicar que uma variável tipo apontador refere-se a um

de menu.

O nono parâmetro indica o nome do ícone associado à janela da aplicação. Neste caso, o nome do ícone associado à janela criada é "IconeBasico". O *bitmap* que representa este ícone deve ter sido previamente criado com auxílio de um editor de recursos e declarado, através de um comando específico, no arquivo de recursos da aplicação.

O último parâmetro define o nome da função-ação associada à janela criada.

Depois de se chamar a função de criação da janela, o seguinte trecho de código é encontrado:

```
if( indJan == NULO ){
      return FALSE;            /* janela não pôde ser criada */
}
```

No trecho acima, o valor de retorno da função de criação é testado para verificar se houve algum erro durante a criação da janela. Em caso de falha, o valor retornado pela função de criação de janelas será NULO e a execução da aplicação deverá ter fim. Em caso de sucesso, o valor de retorno corresponde ao identificador da janela criada.

Se o identificador da janela criada for válido, a janela pode ser mostrada para o usuário. O seguinte trecho de código caracteriza essa ação na aplicação exemplo:

```
U_MostraJanela( idJan );
```

A função **U_MostraJanela()** é a função da biblioteca UNIFIC que torna uma janela criada visível para o usuário. O identificador da janela a ser mostrada deve ser passado como parâmetro para esta função. Esta é a última função que caracteriza a fase de criação na aplicação exemplo.

endereço nulo.

5.4.1.3. Processamento dos Eventos

Logo após a fase de criação da aplicação tem início a fase de execução. O trecho de código que caracteriza esta nova fase é apresentado a seguir:

```
U_LacoPrincipal( apEvento );
```

A função **U_LacoPrincipal()** recebe, como parâmetro, o registro do evento destinado à aplicação. O registro do evento é uma variável do tipo EVENTO que possui todas as informações que uma aplicação necessita saber a respeito do evento ocorrido. Essas informações incluem, por exemplo, o identificador da janela onde o evento ocorreu, tipo do evento, localização do apontador, estado do teclado e hora em que o evento ocorreu. Ao todo, onze eventos diferentes podem ser recebidos por uma aplicação UNIFIC (vide seção A1.4 do apêndice A).

5.4.1.4. Definição da Função-ação

A segunda função definida pela aplicação Basico.c é a função **FuncJanPrinc()**. Esta função é a função-ação que foi associada a janela principal da aplicação. O trecho de código onde a função **FuncJanPrinc()** é definida na aplicação exemplo é apresentado a seguir.

```
long FuncJanPrinc( idJan, apEvento )
U_IDJAN      idJan;
U_EVENTO     apEvento;
{
      U_IDJAN      idDlg;

      switch( U_TIPO( apEvento ) ){
            case E_DESTROI:
                  U_DestroiJanela( idJan );
                  break;
            default:
                  return( U_FuncAcDef( apEvento ) );
      }
      return NULL;
}
```

Toda função-ação UNIFIC segue um padrão geral pré-definido. Os parâmetros de entrada deste tipo de função sempre são dois: o identificador da janela onde o evento ocorreu e o registro que contém as informações sobre o evento ocorrido.

O corpo de uma função-ação, normalmente, é composto por um comando *switch* que associa um *case* distinto para cada evento de interesse da janela. Dentro de cada *case* o programador especifica as ações a serem executadas pela aplicação quando o evento for recebido pela janela.

Na aplicação exemplo, a macro TIPO, fornecida pela UNIFIC, é utilizada no comando *switch* para descobrir o tipo do evento ocorrido. O primeiro *case* do comando *switch* indica a ação a ser executada quando o evento E_DESTROI é recebido pela aplicação. Este evento é recebido por aplicação quando o usuário deseja terminar a sua execução. Quando este tipo de evento ocorre, a aplicação exemplo chama a função **U_DestroiJanela()** para realizar as operações necessárias ao término da aplicação. Apenas esse evento é de interesse para a janela principal da aplicação exemplo.

Os demais eventos recebidos pela janela principal são processados pela opção *default* do comando *switch*. Estes eventos são enviados à função **U_FuncAcDef(),** fornecida pelo UNIFIC para processar os eventos recebidos pela aplicação de uma forma *default*. A função **U_FuncAcDef()** ajuda a garantir a uniformidade do *look-and-feel* das aplicações UNIFIC.

5.4.1.5. Definição do Ícone

O segundo arquivo que compõe a aplicação UNIFIC é o arquivo de recursos da aplicação. Nesse arquivo encontra-se a definição do único recurso utilizado pela aplicação Basico.c: o ícone (vide seção 4.5.2.4). O conteúdo do arquivo Basico.REC é apresentado no trecho de código a seguir:

```
ICONE        iconeBasico IcoBasic.bmp
```

O comando disponível na biblioteca UNIFIC para a definição de ícones no arquivo de recursos é o comando ICONE. O primeiro parâmetro deste comando, especificado logo após a palavra chave ICONE, indica o nome que a aplicação usará para se referir ao ícone que está sendo definido. O segundo parâmetro indica o nome do arquivo que contém o *bitmap* a ser utilizado como ícone.

Pode-se observar que, apesar do ícone ter o seu *bitmap* previamente criado por meio de um editor de recursos (vide seção 5.2), ele deverá ser definido através do comando ICONE no arquivo de recursos para que a aplicação possa localizá-lo durante sua execução.

5.4.2. Criando Menus

O segundo exemplo, discutido neste capítulo, mostra como criar uma janela que possui uma barra de menu. A barra de menu da aplicação Menu.c é composta por um menu *dropdown* cujo título é *Help*. O menu *Help*, por sua vez, possui um único ítem cujo título é *About*. A figura 5.5 mostra qual é a saída gerada pela aplicação Menu.c quando executada em um ambiente Windows.

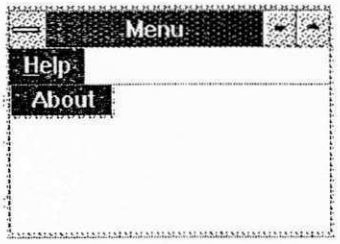

Figura 5.5 Resultado da execução da aplicação Menu.exe em um ambiente Windows.

O código fonte da aplicação Menu.c é apresentado na figura 5.6.

```
/************************************************

Menu.C

************************************************/
#include    "Unific.h"
#include    "Menu.h"

#define     nomeRec     "menu.rec"

int   main();
long  FuncJanPrinc(  U_IDJAN, U_EVENTO );

main()
{
      U_INDJAN    indJan;           /* identificador da janela principal */
      U_EVENTO    apEvento;         /* apontador para estrutura de eventos */

      U_InicializaAplicacao( nomeRec );
      indJan = U_CriaJanela(NULO,          /* identificador do pai da janela */
                  J_JANPRINC | JE_VISIVEL,/* estilo da janela criada */
                  "Menu",                 /* título da janela */
                  CJ_DEFAULT,             /* posição horizontal default */
                  CJ_DEFAULT,             /* posição vertical default */
                  CJ_DEFAULT,             /* largura default */
                  CJ_DEFAULT,             /* altura default */
                  "MenuBasico",           /* nome do menu da janela */
                  "IconeBasico",          /* nome no ícone da janela */
                  FuncJanPrinc            /* função da janela */
                  );
      if( indJan == NULO ){
            return FALSE;                 /* janela não pôde ser criada */
      }

      U_MostraJanela( idJan );

      U_LacoPrincipal( apEvento );

      return TRUE;
}

long FuncJanPrinc( idJan, apEvento )
U_IDJAN     idJan;
U_EVENTO    apEvento;
{
      U_IDJAN     idDlg;

      switch( U_TIPO( apEvento ) ){
            case E_COMANDO:
                  if( U_ID( apEvento ) == ID_ABOUT){
                  }
                        U_QuadroMensagem( idJan, "Característica não
                                    implementada", "About", QD_OK );
                  else{
                        return( U_FuncAcDef( apEvento ) );
                  }
            case E_DESTROI:
                  U_DestroiJanela( idJan );
                  break;
```

```
            default:
                  return( U_FuncAcDef( apEvento ) );
      }
      return NULL;
}

/*************************************************

Menu.H

*************************************************/

#define           ID_ABOUT          100
#define           ID_HELP           101

/*************************************************

Menu.REC

*************************************************/
ICONE       iconeBasico IcoBasic.bmp

MENU MenuBasico
{
      SUBMENU     "&Help"      ID_HELP
      {
            ITEM  "About..."   ID_ABOUT
      }
}
```

Figura 5.6 Aplicação Menu.c.

5.4.2.1. Especificação do menu

A primeira diferença entre a aplicação Basico.C e a aplicação Menu.c pode ser encontrada na definição do oitavo parâmetro da função **U_CriaJanela()**. A definição desse valor pode ser verificada no trecho de código apresentado a seguir:

```
      indJan = U_CriaJanela(  NULO, /* identificador do pai da janela */
                  J_JANPRINC,       /* estilo da janela principal */
                  "Menu",           /* título da janela */
                  CJ_DEFAULT,       /* posição horizontal default */
                  CJ_DEFAULT,       /* posição vertical default */
                  CJ_DEFAULT,       /* largura default */
                  CJ_DEFAULT,       /* altura default */
                  "MenuBasico",     /* nome do menu da janela */
                  "IconeBasico",    /* nome no ícone da janela */
                  FuncJanPrinc      /* função da janela */
                  );
```

O oitavo parâmetro, nesse caso, é definido como a cadeia de caracteres "MenuBasico". Isto significa que este é o nome da barra de menu da janela criada. Esta é a forma que um programador dispõe para associar uma barra de menu a uma janela. O menu especificado deve ter sido previamente definido no arquivo de recursos da aplicação.

5.4.2.2. Ações Associadas ao Menu

Na função-ação da janela principal, um novo trecho de código foi acrescentado. A listagem do código com este novo trecho é apresentado a seguir:

```
long FuncJanPrinc( idJan, apEvento )
U_IDJAN      idJan;
U_EVENTO     apEvento;
{
      U_IDJAN     idDlg;

      switch( U_TIPO( apEvento ) ){
            case E_COMANDO:
                  if( U_ID( apEvento ) == ID_ABOUT){
                        U_QuadroMensagem( idJan, "Característica não
                                          implementada", "About", QD_OK ):
                  else{
                        return( U_FuncAcDef( apEvento ) );
                  }
                  break;
            case E_DESTROI:
                  U_DestroiJanela( idJan );
                  break;
            default:
                  return( U_FuncAcDef( apEvento ) );
      }
      return NULL;
}
```

O techo de código acrescentado à função **FuncJanPrinc()** indica que a ocorrência do evento E_COMANDO agora é importante para a janela principal. Esse tipo de evento é recebido pela aplicação quando o usuário ativa um dos itens de um menu da aplicação. Quando esse evento é recebido pela função **FuncJanPrinc()**, a macro ID da biblioteca UNIFIC é utilizada para obter

o código associado ao ítem que foi ativado. Em seguida, a função verifica se o item ativado está associado ao código ID_ABOUT (o que significa que o usuário ativou o item *About* do menu *Help* - para maiores explicações veja a próxima seção). Caso afirmativo, a função **U_QuadroMensagem()** da UNIFIC é chamada para apresentar ao usuário um quadro com a mensagem "Característica não implementada". Caso negativo o evento será processado pela função **U_FuncAcDef()**.

5.4.2.3. Definição do Menu

Já no arquivo de recursos, pode-se verificar que um novo comando foi acrescentado: o comando MENU. O trecho de código que representa o novo comando é apresentado a seguir:

```
MENU MenuBasico
{
        SUBMENU     "&Help"      ID_HELP
        {
              ITEM  "About..."   ID_ABOUT
        }
}
```

O comando MENU é utilizado para definir um menu no arquivo de recursos. O primeiro parâmetro especificado, logo após a palavra chave MENU, indica o nome que a aplicação utilizará para referenciar menu que está sendo definido. No exemplo acima, o nome do menu que está sendo definido é "MenuBasico". Em seguida, o subcomando SUBMENU é utilizado para indicar que o menu definido é composto por um submenu chamado *Help*. O '&' especificado na cadeia de caracteres *Help* indica que a letra H é a *hotkey* do item definido. Dentro do comando submenu, o comando ITEM é utilizado para definir o único item que compõe o menu *Help*, o item *About*. As constantes ID_HELP e ID_ABOUT definem os respectivos códigos que devem ser enviados para a aplicação quando algum dos itens for ativado pelo usuário.

5.4.3. Criando Quadros de Diálogo

O terceiro exemplo apresentado neste capítulo mostra como criar um quadro de diálogo. O quadro de diálogo criado pela aplicação Dialogo.c fornece informações a respeito do nome e da versão da aplicação desenvolvida. A figura 5.7 mostra a saída gerada pela aplicação quando está sendo executada em um ambiente Windows.

Figura 5.7 Resultado da execução da aplicação Dialogo.exe em um ambiente Windows.

O código fonte da aplicação Dialogo.c é apresentado na figura 5.8.

```
/************************************************

Dialogo.C

************************************************/
#include    "Unific.h"
#include    "Dialogo.h"

#define     nomeRec     "dialogo.rec"

int   main();
long  FuncJanPrinc(  U_IDJAN, U_EVENTO );
BOOL  FuncJanDlg( U_IDJAN, U_EVENTO );

main()
{
      U_INDJAN    indJan;     /* identificador da janela principal */
      U_EVENTO    apEvento;   /* apontador para a estrutura de eventos */

      U_InicializaAplicacao( nomeRec );
      indJan = U_CriaJanela(  NULO,       /* identificador do pai da janela */
                  J_JANPRINC | JE_VISIVEL, /* estilo da janela criada */
                  "Dialogo",              /* título da janela */
                  CJ_DEFAULT,             /* posição horizontal default */
                  CJ_DEFAULT,             /* posição vertical default */
                  CJ_DEFAULT,             /* largura default */
                  CJ_DEFAULT,             /* altura default */
```

```
                    "MenuBasico",           /* nome do menu da janela */
                    "IconeBasico",          /* nome no ícone da janela */
                    FuncJanPrinc            /* função da janela */
                    );
      if( indJan == NULO ){
            return FALSE;                   /* janela não pôde ser criada */
      }

      U_MostraJanela( idJan );

      U_LacoPrincipal( apEvento );


      return TRUE;
}

long FuncJanPrinc( idJan, apEvento )
U_IDJAN     idJan;
U_EVENTO    apEvento;
{
      U_IDJAN     idDlg;

      switch( U_TIPO( apEvento ) ){
            case E_COMANDO:
                  if( U_ID( apEvento ) == ID_ABOUT){
                        idDlg = U_CriaDialogo( "DlgBasico",/* nome do Q.D. */
                                          idJan,          /* pai do Q.D. */
                                          FuncJanDlg      /*função do Q.D. */
                                          );
                  }
                  else{
                        return( U_FuncAcDef( apEvento ) );
                  }
            case E_DESTROI:
                  U_DestroiJanela( idJan );
                  break;
            default:
                  return( U_FuncAcDef( apEvento ) );
      }
      return NULL;
}

BOOL FuncJanDlg( idDlg, apEvento )
U_IDJAN     idDlg;
U_EVENTO    apEvento;
{
      switch( U_TIPO( apEvento ) ){
            case E_COMANDO:
                  if( U_ID( apEvento ) == ID_OK ){
                        U_DestroiDialogo( idDlg );
                        return TRUE;
                  }
                  break;
      }
      return FALSE;
}

/*****************************************************
```

Dialogo.H

```
************************************************/

#define          ID_ABOUT          100
#define          ID_HELP           101

#define          ID_OK             200

/************************************************

Dialogo.REC

************************************************/
ICONE       iconeBasico IcoBasic.bmp

MENU MenuBasico
{
      SUBMENU      "&Help"      ID_HELP
      {
            ITEM   "About..."   ID_ABOUT
      }
}

DIALOGO     DlgBasico   22 17 144 75       JD_MODO
TITULO      "Dialogo Basico"
{
      JC_ESTATICO -1      0,   5, 144,  8, "Biblioteca UNIFIC"      CS_CENTR
      JC_ESTATICO -1      0,  14, 144,  8, "Aplicação Básica"       CS_CENTR
      JC_ESTATICO -1      0,  34, 144,  8, "Versão 1.0"             CS_CENTR
      JC_COMAND   ID_OK 53,  59,  32, 14, "Ok"                      CC_DEFCOMAND
}
```

Figura 5.8 Aplicação Dialogo.c

5.4.3.1. Criação do Quadro de diálogo

A primeira diferença significativa entre a aplicação Dialogo.c e a aplicação Menu.c encontra-se na função-ação da janela principal, apresentada no trecho de código a seguir:

```
long FuncJanPrinc( idJan, apEvento )
U_IDJAN     idJan;
U_EVENTO    apEvento;
{
      U_IDJAN     idDlg;

      switch( U_TIPO( apEvento ) ){
            case E_COMANDO:
                  if( U_ID( apEvento ) == ID_ABOUT){
                        idDlg = CriaDialogo( "DlgBasico",/*nome do Q.D.*/
                                             idJan,      /*pai do Q.D.*/
                                             FuncJanDlg  /*função do Q.D.*/
                                           );
                  }
```

```
                else{
                        return( U_FuncAcDef( apEvento ) );
                }
        case E_DESTROI:
                U_DestroiJanela( idJan );
                break;
        default:
                return( U_FuncAcDef( apEvento ) );
    }
    return NULL;
}
```

Quando evento E_COMANDO é recebido pela aplicação Dialogo.c e constata-se que este evento foi causado pela ativação do item *About* do menu *Help*, a função **U_CriaDialogo()** é chamada. A função **U_CriaDialogo()** é a função oferecida pela biblioteca UNIFIC para criar quadros de diálogo. Essa função recebe como parâmetros o nome associado ao quadro de diálogo no arquivo de recursos, o identificador da janela pai do quadro de diálogo e o nome da função-ação do quadro de diálogo. Os demais atributos do quadro de diálogo devem ter sido definidos pelo programador no arquivo de recursos da aplicação (veja seções 4.5.2.3 e 5.4.3.3). No exemplo acima o nome associado ao quadro de diálogo a ser criado é "DlgBasico", a janela definida como pai do quadro de diálogo é a janela principal da aplicação e o nome da função-ação associada ao quadro de diálogo é **FuncJanDlg()**.

5.4.3.2. Função-ação do Quadro de Diálogo

Ainda no arquivo Dialogo.c pode-se observar que uma nova função-ação foi definida, a função **FuncJanDlg()**. Esse é o nome da função associada ao quadro de diálogo, através da função de criação do quadro de diálogo. O trecho de código a seguir mostra a definição dessa função:

```
BOOL FuncJanDlg( idDlg, apEvento )
U_IDJAN     idDlg;
U_EVENTO    apEvento;
{
        switch( U_TIPO( apEvento ) ){
                case E_COMANDO:
```

```
                    if( U_ID( apEvento ) == ID_OK ){
                        U_DestroiDialogo( idDlg );
                        return TRUE;
                    }
                    break;
            }
            return FALSE;
    }
```

A única diferença entre a função-ação definida para uma janela comum e a função-ação definida para um quadro de diálogo é o seu valor de retorno. Enquanto a função-ação da primeira retorna um *long*, o retorno da segunda é um booleano. Os parâmetros de entrada e a estrutura geral da função são idênticos nos dois casos. Na função **FuncJanDlg(),** o único evento especificado como evento de interesse para o quadro de diálogo é o evento E_COMANDO. Esse tipo de evento é recebido por um quadro de diálogo quando algum de seus controles é ativado pelo usuário. Na aplicação exemplo, quando esse tipo de evento ocorre, a função-ação do quadro de diálogo verifica se o botão de comando "Ok" do quadro de diálogo foi ativado. Isto ocorre apenas quando o código ID_OK é especificado no evento. Caso afirmativo, a função **U_DestroiDialogo()** da biblioteca UNIFIC é utilizada para destruir o quadro de diálogo.

5.4.3.3. Definição do Quadro de Diálogo

No arquivo de recursos da aplicação Dialogo.c, um novo comando foi acrescentado: o comando DIALOGO. O trecho de código que representa esse comando é apresentado a seguir:

```
DIALOGO     DlgBasico   22, 17, 144, 75,  JD_MODO
TITULO      "Dialogo Basico"
{
     JC_ESTATICO  -1    0,  5, 144,  8,  "Biblioteca UNIFIC"     CS_CENTR
     JC_ESTATICO  -1    0, 14, 144,  8,  "Aplicação Básica"      CS_CENTR
     JC_ESTATICO  -1    0, 34, 144,  8,  "Versão 1.0"            CS_CENTR
     JC_COMAND   ID_OK 53, 59,  32, 14,  "Ok"                    CC_DEFCOMAND
}
```

O comando DIALOGO no arquivo de recursos indica a definição de um quadro de diálogo. O nome do quadro de diálogo deve ser especificado logo após a palavra chave DIALOGO. No exemplo acima, o quadro de diálogo chama-se "DlgBasico". Em seguida devem ser especificados os parâmetros x, y, cx, cy que definem a coordenada do canto superior esquerdo e a coordenada do canto inferior direito do quadro de diálogo. O último parâmetro da primeira linha do comando DIALOGO indica o modo do quadro de diálogo. A constante JD_MODO indica que o quadro de diálogo definido é um quadro de diálogo com modo na aplicação.

O comando TITULO é um subcomando opcional do comando DIALOGO que permite ao programador definir o título a ser apresentado na borda do quadro de diálogo. No exemplo discutido, o título do quadro de diálogo é "Dialogo Basico" .

Depois que as características gerais do quadro de diálogo foram especificadas, os controles que compõem o quadro de diálogo podem ser definidos. No exemplo acima, quatro controles compõem o quadro de diálogo: três textos estáticos e um botão de comando.

O comando JC_ESTATICO é oferecido pela biblioteca UNIFIC para que o programador defina os controles do tipo texto estático. O primeiro parâmetro desse comando, que deve ser especificado logo após a palavra chave JC_ESTATICO, indicará o código associado ao controle criado. O código -1 é associado aos três textos estáticos no exemplo discutido. Nesse caso, o código associado aos textos estáticos do quadro de diálogo são indiferentes para a aplicação, pois este tipo de controle não pode ser ativado pelo usuário.

Depois da definição do identificador do controle, os seus parâmetros de localização devem ser especificados. Estes parâmetros são definidos em relação à área interna da janela do quadro de diálogo.

O sexto parâmetro do comando JC_ESTATICO define o título do controle. No exemplo

apresentado acima os três textos estáticos apresentam os títulos "Biblioteca UNIFIC", "Aplicação Básica" e "Versão 1.0", respectivamente.

O último parâmetro do comando JC_ESTATICO define os atributos do texto estático. O atributo CS_CENTR indica que os três textos estáticos devem estar centralizados.

O comando JC_COMAND é o comando oferecido pela biblioteca UNIFIC para a definição dos botões de comando. O primeiro parâmetro depois da palavra chave JC_COMAND indica o código associado ao botão de comando. No exemplo acima a constante ID_OK é definida como o código do controle criado. Em seguida são definidos os parâmetros de localização do botão de comando. Depois que a localização do controle foi definida, a cadeia de caracteres "Ok" indica o título que será apresentado pelo botão de comando. Finalmente o atributo CC_DEFCOMAND indica que o botão de controle definido é o botão de comando *default* do quadro de diálogo.

Uma vez que as características de programação da biblioteca UNIFIC foram apresentadas, é necessário comprovar a viabilidade desta biblioteca através da sua implementação em algum ambiente gráfico. Os detalhes da implementação da biblioteca UNIFIC, no ambiente Windows, é o assunto a ser discutido no próximo capítulo.

Capítulo 6

6. Implementação da Biblioteca UNIFIC

Este capítulo mostra os detalhes da implementação da biblioteca UNIFIC, cuja especificação foi feita no capítulo 4. Inicialmente serão apresentados os módulos funcionais que compõem a biblioteca UNIFIC e descritas, em linguagem "C", as estruturas de dados necessárias para o funcionamento de cada um destes módulos. Finalmente serão discutidos os principais aspectos da implementação da UNIFIC, no ambiente Windows, enfatizando-se como algumas características particulares deste ambiente influenciaram nas soluções adotadas para esta implementação.

6.1. Visão Geral dos Módulos Componentes

A partir do modelo de programação unificado definido no capítulo 4, a biblioteca UNIFIC pode ser projetada como um conjunto integrado de cinco módulos. Na implementação da primeira versão da bibloteca UNIFIC são colocados à disposição do programador os quatro módulos mais importantes: serviços do sistema operacional, entrada de dados orientada a eventos, saída gráfica e objetos da GUI. O quinto módulo, o de troca de dados, apesar de fazer parte do modelo de programação unificado definido no capítulo 4, estará disponível somente em futuras versões da biblioteca UNIFIC. É importante observar que os módulos da API UNIFIC devem ser implementados em cada ambiente gráfico unificado, considerando as particularidades de cada um dos ambientes. A funcionalidade dos módulos que compõem a API UNIFIC, bem como as

estruturas de dados utilizadas, serão analisadas a seguir.

6.1.1. Módulo de Serviços do Sistema Operacional

O módulo de serviços do sistema operacional implementa as funções que permitem a interação entre as aplicações UNIFIC e o sistema operacional do ambiente destino. As funções disponíveis classificam-se em: entrada e saída em arquivos, execução de aplicações e gerência de memória. Nenhuma estrutura de dados específica foi necessária durante a implementação das funções desse módulo. Isto foi possível porque o estado de todas as operações oferecidas pelo módulo de serviços do sistema operacional não precisa ser mantido pela UNIFIC, mas apenas pelo sistema operacional onde as operações serão realizadas.

6.1.2. Módulo da Entrada de Dados Orientada a Eventos

O módulo de entrada de dados orientado a eventos implementa os serviços que estão relacionados com o envio e recebimento de eventos pelas aplicações UNIFIC. Para implementar este tipo de serviço é necessário que a UNIFIC mantenha informações a respeito do evento a ser enviado ou recebido pela aplicação. Estas informações são, normalmente, armazenadas em um registro de eventos, como foi discutido na seção 2.3.3.1. As características do registro de eventos adotado pela biblioteca UNIFIC serão apresentadas a seguir.

6.1.2.1. Registro de Eventos UNIFIC

O registro de eventos da UNIFIC armazena informações sobre onze diferentes tipos de eventos externos (vide seção 4.3.2). Os eventos são gerados originalmente pela GUI do ambiente onde a aplicação está sendo executada. Em seguida, antes que a aplicação possa recebê-lo, a biblioteca UNIFIC o intercepta e o traduz para um registro de evento UNIFIC. A seguir será

descrito o conteúdo do registro de eventos UNIFIC, de acordo com a numeração apresentada à esquerda do código:

```
typedef struct{

(1)   U_IDJAN     jan;              /* únicos atributos dos eventos */
(2)   int         tipo;             /*  E_CRIA, E_DESTROI e E_SAI */

(3)   union{      /* atributos adicionais para os eventos do tipo: */

(3.1)       struct{                 /* E_MOUSE_DUPLO,  E_MOUSE_LIBERA,
                                       E_MOUSE_PRESS, E_MOUSE_MOVE */
                  E_RET localizacao;/* localização do mouse */
                  BOOL  shift;      /* estado do shift do teclado */
                  BOOL  controle;   /* estado do ctrl do teclado */
                  BOOL  alt;        /* estado da tecla Alt no teclado */
                  short tecla;      /* código da tecla pressionada */
            }mouse;

(3.2)       struct{                 /* E_CARAC */
                  short ch;         /* código da tecla pressionada */
                  BOOL  shift;      /* estado da tecla shift */
                  BOOL  controle;   /* estado da tecla ctrl */
                  BOOL  alt;        /* estado da tecla alt */
            }carac;

                                    /* E_FOCO
(3.3)       BOOL  foco;             obtem foco = TRUE; libera foco = FALSE */

                                    /* E_RESTAURA     */
(3.4)       E_RET restaura;         /* área da janela a ser restaurada */

                                    /* E_COMANDO      */
(3.5)       struct{                 /* ativação de item ou de controle */
                  IDMENU      id;   /* ID do menu ou quadro de diálogo */
                  BOOL        shift;/* estado do shift do teclado */
                  BOOL        controle;/* estado do ctrl do teclado */
            }comando;

      }atrib;

}E_EVENTO;
```

(1) janela: indica o identificador da janela a qual o evento está associado;

(2) tipo: indica o tipo do evento recebido pela aplicação. Os eventos podem ser do tipo: E_DESTROI, E_CRIA, E_SAI, E_MOUSE_DUPLO, E_MOUSE_LIBERA, E_MOUSE_PRESS, E_MOUSE_MOVE, E_CARAC, E_FOCO, E_RESTAURA e E_COMANDO. Maiores informações sobre estes eventos podem ser encontradas na seção A1.4;

(3) atributos adicionais: informações adicionais necessárias para alguns tipos de eventos:

(3.1) mouse: atributos relacionados aos eventos que indicam a interação entre o usuário e o *mouse* - eventos E_MOUSE_DUPLO, E_MOUSE_LIBERA, E_MOUSE_PRESS, E_MOUSE_MOVE. Para este tipo de evento é necessário saber a posição do *mouse* na hora do evento, o estado das teclas *Shift, Ctrl* e *Alt*, e ainda o código da tecla do *mouse* pressionada (centro, esquerda ou direita);

(3.2) caractere: atributos relacionados ao evento que indica a interação do usuário com o teclado - evento E_CARAC. Para este tipo de evento as informações armazenadas são o estado das teclas *Shift, Ctrl* e *Alt*, além do código da tecla pressionada;

(3.3) foco: atributo relacionado ao evento E_FOCO. Este atributo indica se a janela associada ao evento deve receber ou perder o foco de entrada[32];

(3.4) restaura: atributo relacionado ao evento que indica a existência de uma área a ser restaurada na janela - evento E_RESTAURA. As coordenadas da área inválida são fornecidas como atributo desse tipo de evento;

(3.5) comando: atributos relacionados ao evento que indica a ativação de um item de menu ou de um controle de quadro de diálogo - evento E_COMANDO. Para este tipo de evento são fornecidas informações sobre o código do item ou controle ativado, e o estado das teclas *Shift* e *Ctrl* no teclado.

6.1.3. Módulo de Saída Gráfica

O módulo de saída gráfica implementa os serviços relacionados à geração de saída gráfica das aplicações UNIFIC. A primeira versão da biblioteca UNIFIC coloca, à disposição do programador, funções para a criação de formas geométricas e escrita de textos. As funções deste

[32] O termo foco de entrada refere-se ao objeto da GUI que recebe os eventos provenientes do *mouse* ou teclado.

módulo que necessitam da definição de atributos gráficos adotam os atributos *defaults* do ambiente destino. Como não é necessário que a UNIFIC mantenha informações de estado durante a execução das operações oferecidas por este módulo na primeira versão, nenhuma estrutura de dados especial foi utilizada.

6.1.4. Módulo dos Objetos da Interface do Usuário

O objetivo deste módulo é oferecer serviços que permitam ao programador criar e gerenciar os objetos da GUI disponíveis na UNIFIC. Nesse módulo, dois objetos se destacam devido à sua importância: janela e menu. Para manter informações a respeito desses objetos, duas estruturas de dados foram utilizadas: tabela de janelas e tabela de menus. As características de ambas serão discutidas a seguir:

6.1.4.1. Tabela de Janelas

Quando uma aplicação UNIFIC cria uma janela, todas as informações relativas a esta janela são armazenadas em uma tabela de janelas (Figura 6.1). Essa tabela mantém informações atualizadas sobre as janelas criadas pelas aplicações UNIFIC.

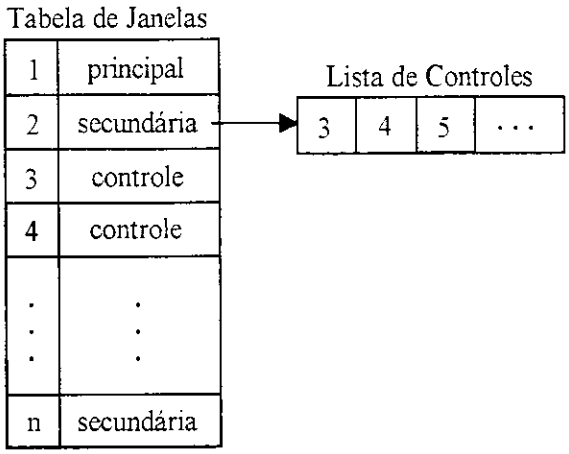

Figura 6.1 Tabela de janelas

Duas classes de informações são mantidas para cada janela criada: gerais e específicas.

• **Informações Gerais**: são informações de interesse para qualquer janela, independentemente do seu tipo. O código apresentado a seguir mostra o registro de informações gerais, especificado em linguagem "C", a respeito de um elemento da tabela de janelas:

```
typedef struct {

(1)    BOOL        janFixa;          /* permanência das informações */
(2)    BOOL        janValid;         /* validade das informações */
(3)    U_JFLAG     janTipo           /* tipo da janela */
(4)    char        [janTitMAXNOME];  /* título da borda da janela */

(5)    struct{                       /* área da janela */
            int    ix;
            int    iy;
            int    cx;
            int    cy;
       }janRet;

(6)    U_APJAN     apJan;            /* ID da janela na GUI destino */
(7)    int         indPai;           /* pai da janela */
(8)    U_JFLAG     janEstado;        /* estado da janela */
(9)    U_JFLAG     janBorda;         /* tipo da borda */

}E_GERAL;
```

(1) fixa: indica se as informações da janela devem permanecer na tabela de janelas mesmo que a janela tenha sido destruída. As informações da janela serão mantidas na tabela de janelas independentemente de sua existência, apenas quando a janela for criada a partir das informações do arquivo de recursos. Isso evita que as informações tenham que ser recarregadas do arquivo de recursos sempre que a janela for recriada pela aplicação. Se, por outro lado, a janela for criada através da função de criação de janelas **U_CriaJanela()**, quando a janela for destruída todas as suas informações são invalidadas e o seu espaço na tabela pode ser preenchido com informações de outra janela;

(2) validade: indica se a janela está sendo utilizada pela aplicação;

(3) tipo: indica o tipo da janela. Uma janela UNIFIC pode ser de três tipos: principal,

secundária e de controle;

(4) título: indica o título apresentado na borda da janela;

(5) área da janela: indica a localização e o tamanho da janela;

(6) identificador da janela na GUI destino: referencia a janela criada na GUI destino;

(7) pai: indica qual o identificador do pai da janela;

(8) estado: indica o estado em que se encontra a janela. Uma janela UNIFIC pode apresentar-se em três estados durante a execução da aplicação: visível, inativa, maximizada ou minimizada;

(9) tipo da borda: indica o tipo da borda da janela. A borda da janela pode ser de três tipos: redimensionável, dupla ou simples.

• **Informações Específicas**: este tipo de informação varia de acordo com o tipo da janela. Os três trechos de código apresentados a seguir mostram os registros de informações específicas para as janelas principais, secundárias e de controle, respectivamente:

Trecho 1: informações específicas para as janelas principais.

```
typedef struct{

(1)   struct{                          /* atributos da borda */
            BOOL  menuSist;            /* existência de menu do sistema */
            BOOL  maxIco;              /* ícone para maximização da borda */
            BOOL  minIco;              /* ícone para minimização da borda */
            BOOL  bordaTit;            /* barra de título */
            BOOL  hRolam;              /* barra de rolamento horizontal */
            BOOL  vRolam;              /* barra de rolamento vertical */
      }modifPrinc;

(2)   int          idMenu;             /* identificador do menu */
(3)   int          idIco;              /* identificador do ícone */
(4)   U_APPROC     janFunc;            /* função-ação */

}princip;
```

O registro específico de uma janela principal contém:

(1) modificadores da janela principal: indicam os atributos presentes na borda da janela principal. Esses atributos definem a existência do menu do sistema, ícone de maximização, ícone de minimização, barra de título, barra de rolamento vertical e barra de rolamento horizontal;

(2) menu: indica o nome associado à barra de menu da janela;

(3) ícone: indica o nome do ícone associado à janela;

(4) função-ação: indica o endereço da função-ação associada à janela.

Trecho 2: informações específicas para as janelas secundárias.

```
struct{
      (1)   char  nomeSec[ MAXNOME ];      /* nome associado a janela */

      (2)   struct{                        /* atributos da borda */
                  BOOL  menuSist;          /* menu do sistema */
                  BOOL  maxIco;            /* ícone de maximização */
                  BOOL  minIco;            /* ícone de minimização */
                  BOOL  bordaTit;          /* barra de título */
                  BOOL  hRolam;            /* barra de rolamento horizontal */
                  BOOL  vRolam;            /* barra de rolamento vertical */
            }modifSec;

      (3)   U_LCONTR    listaIdContr;      /* lista de controles */
      (4)   U_FARPROC   dlgFunc;           /* função-ação */
}secund;
```

O registro específico das janelas secundárias, por sua vez, contém:

(1) nome: identificador, numérico ou alfabético, associado a uma janela secundária, durante a definição de um quadro de diálogo no arquivo de recursos;

(2) modificadores da janela secundária: estes atributos são idênticos aos definidos para a janela principal;

(3) lista de controles: para as janelas secundárias utilizadas como quadros de diálogo este parâmetro indica o endereço da lista de identificadores dos controles que compõe o quadro de diálogo;

(4) função-ação: indica o endereço da função-ação associada à janela secundária.

Trecho 3: Informações específicas para as janelas de controle:

```
typedef struct{

(1)   U_JFLAG     contrTipo;              /* tipo do controle */
(2)   int         contrId;                /* identificador do controle */

(3)   union{

                                /* atributos de botão de comando */
(3.1)       BOOL  comandDef;    /* botão de comando default */


(3.2)       struct{ /* atributos de botão de radio ou quadro de marcar */
                  int         grupoId;    /* identificador do grupo */
                  BOOL        marcado;    /* estado do controle */
            }grupo;

                                          /* atributos de texto estatico */
(3.3)       U_JFLAG     estaticJustif;    /* alinhamento do texto */

(3.4)       struct{ /* atributos de texto editável */
                  BOOL        multilinha; /* entrada multilinha */
                  U_JFLAG     tipoEntrada;/* conversão dos caracteres */
            }edicao;

(3.5)       struct{ /* atributos de quadro de lista */
                  BOOL        multisel;   /* seleção múltipla */
                  BOOL        alfab;      /* ordem alfabética */
             }lista;

      }atribContr;   /* union */

}contr;
```

As informações específicas mantidas para as janelas de controle são:

(1) tipo: indica o tipo da janela de controle. Uma janela de controle pode ser do tipo: botão de controle, botão de rádio, quadro de marcar, texto estático, quadro de texto editável ou quadro de lista;

(2) código do controle: valor numérico que identifica o controle perante à janela a qual o controle pertence;

(3) atributos do controle: especificam as características particulares de alguns tipos de controle:

(3.1) comando: indica o tipo do botão de comando. Um botão de comando pode ser definido como o botão de comando *default* do quadro de diálogo;

(3.2) grupo: indica se o botão de rádio ou quadro de marcar pertence a um determinado grupo ou está marcado;

(3.3) texto estático: indica o tipo de alinhamento de um texto estático. Um texto estático pode ser alinhado à esquerda, à direita ou centralizado;

(3.4) texto editável: indica o número de linhas e o tipo da entrada de um texto editável. Um texto editável pode ter uma ou várias linhas de edição e ainda transformar automaticamente todos os caracteres de entrada, digitados através do teclado, em letras maiúsculas ou minúsculas;

(3.5) lista: indica a ordenação e tipo de seleção de um quadro de lista. Um quadro de lista pode ordenar automaticamente suas opções em ordem alfabética e permitir que mais de uma das suas opções sejam selecionadas pelo usuário.

6.1.4.2. Tabela de Menus

A segunda estrutura de dados, necessária ao funcionamento do módulo dos objetos da GUI, é a tabela de menus. A tabela de menus mantém informações a respeito de todos os menus criados pelas aplicações UNIFIC (Figura 6.2).

Cada elemento da tabela de menus é um registro que contém informações a respeito de um determinado menu. Dentre todas as informações armazenadas no registro do menu, a que mais se destaca é o endereço da lista de itens que compõe este menu (figura 6.2a). Da mesma forma, cada item do menu que ativar um submenu, aponta para o endereço da lista de itens correspondente ao submenu (figura 6.2b). De forma análoga, cada item do submenu que ativar outro submenu, aponta para o endereço do submenu correspondente (figura 6.2c).

Como todos os menus de uma aplicação UNIFIC são definidos no arquivo de recursos

(vide seção 4.5.2.2), as informações referentes aos menus de uma aplicação são carregadas, durante a fase de inicialização da aplicação, para a tabela de menus. Estas informações permanecem na tabela de menus até o término da execução da aplicação, quando então são descartadas.

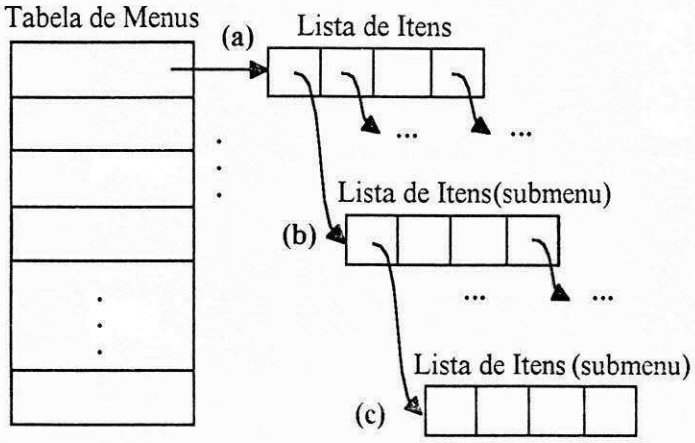

Figura 6.2 Tabela de Menus

O registro apresentado a seguir mostra as informações mantidas na tabela de menus para um menu UNIFIC:

```
typedef     struct{

(1)   char        nomeMenu[ MAXNOME ];                  /* nome do menu */
(2)   U_JFLAG     estado;                               /* estado do menu */

(3)   struct      mitens{/* apontador para a lista de itens */
            int   itemTipo;                             /* tipo do item */

(3.1)       struct{     /* atributos dos itens comando, opção ou submenu */
                  unsigned    cod;              /* identificador do iter */
                  char        titulo[ MAXNOME ];      /* título do item */
                  char        hotkey;                 /* hotkey do item */
                  BOOL        ativo;                  /* estado do item */

                  union{
(3.2)                             /* atributo de item tipo opção */
                        BOOL  opcChecado;             /* estado  */

(3.3)                   struct{     /* atributos dos itens submenus */
                              struct mitens     *submenu;/* submenu */
                              U_APMENU          handleSubWin;/* GUI destino */
                        }sub;
                  }opcsub;

            }itemMenu;
```

```
            struct mitens      *proxItem;              /* próximo item */
        }*listaItens;
(4)     U_APMENU     handleWin;                    /* menu na GUI destino */
}E_MENU;
```

(1) nome: nome associado ao menu, durante a sua definição no arquivo de recursos;

(2) estado: indica se o menu está ativo ou não;

(3) lista de itens: indica o endereço da lista de itens do menu. Os itens dessa lista podem ser do tipo: separador, comando, opção ou submenu:

(3.1) atributos gerais para os itens do tipo comando, opção ou submenu: para esses tipos de itens armazenam-se informações sobre o identificador do item, seu título, sua *hotkey* e seu estado (ativo ou desativo);

(3.2) opção: indica os atributos específicos de um item de menu do tipo opção. Se o item for do tipo opção, além dos atributos gerais é necessário saber se o item está marcado ou não;

(3.3) submenu: indica os atributos específicos de um item de menu do tipo submenu. Se o item for do tipo submenu, além dos atributos gerais, armazenam-se o apontador da lista de itens que compõe o submenu e identificador deste submenu na GUI destino;

(4) identificador do menu na GUI destino: referencia o menu criado na GUI destino.

6.2. Suporte UNIFIC para Ambiente Windows

Para mostrar, na prática, a viabilidade da biblioteca UNIFIC, os detalhes da sua implementação para o ambiente Windows serão discutidos nesta seção. No Windows, a biblioteca UNIFIC é uma biblioteca de ligação dinâmica[33]. Dessa forma, todas as aplicações UNIFIC

[33] Uma biblioteca de ligação dinâmica é uma biblioteca cujo ligação entre a biblioteca e a aplicação é feita apenas em tempo de execução. A maior vantagem desse tipo de biblioteca é permitir que várias aplicações compartilhem uma única cópia das rotinas da biblioteca.

executadas em ambiente Windows compartilham o mesmo código da biblioteca UNIFIC. Isso se traduz em uma grande economia de memória, uma vez que, à medida que o número de aplicações UNIFIC aumentar no ambiente Windows, o código da biblioteca não será replicado.

Essa versão da biblioteca UNIFICfoi desenvolvida em um PC-486 de 25MHz, 8M de RAM, placa Super VGA e com 160MB de disco. O ambiente operacional utilizado foi o Microsoft Windows 3.1 e DOS 6.0. A ferramenta de programação utilizada foi o compilador Borland C/C++ para Windows, versão 3.1.

A primeira versão da biblioteca UNIFIC em ambiente Windows foi implementada com o objetivo de validar as principais características da API UNIFIC. A implementação envolveu os seguintes módulos básicos: o módulo de objetos da GUI, o módulo de entrada de dados orientado a eventos e o módulo de serviços do sistema operacional. O módulo de objetos da GUI foi implementado totalmente. Os módulos de entrada de dados orientado a eventos e o de serviços do sistema operacional foram implementados apenas parcialmente porque nem todas as características oferecidas por estes módulos foram necessárias para a realização da validação desejada.

Durante essa implementação, várias decisões de projeto particulares precisaram ser tomadas. A seguir, serão analisados alguns aspectos da implementação que se relacionam com: inicialização da aplicação, alocação de memória dinâmica, processamento da entrada de dados e criação dos objetos da GUI, da biblioteca UNIFIC no ambiente Windows.

6.2.1. Inicialização da Aplicação

A função de inicialização da biblioteca UNIFIC deve executar as seguintes tarefas: habilitar a aplicação UNIFIC para ser executada no ambiente destino e carregar o arquivo de recursos.

6.2.1.1. Habilitar uma Aplicação para Windows

Para que qualquer aplicação seja executada no ambiente Windows é necessário que esta aplicação obtenha do gerente de aplicações Windows, três parâmetros: instância da aplicação, instância prévia da aplicação e o estado inicial da janela principal da aplicação.

A **instância da aplicação** é um código numérico único que identifica a aplicação perante o Windows.

A **instância prévia** indica a existência de outra instância da mesma aplicação sendo executada no Windows. Este parâmetro vai permitir o compartilhamento de código e recursos entre aplicações afins.

O **estado inicial** da janela principal indica se a janela principal da aplicação vai se apresentar aberta ou iconizada quando a aplicação for carregada.

Para as aplicações Windows, estes parâmetros são fornecidos automaticamente. Mas como obter estes parâmetros para uma aplicação "C" comum, como é o caso da aplicação UNIFIC? A obtenção desses parâmetros foi possível através da implementação de uma função especial, escrita em assembler, que interage com o gerente de aplicações Windows, obtém os parâmetros necessários e os coloca, de forma correta, na pilha da aplicação. Maiores informações sobre a implementação dessa função podem ser encontradas na referência [Mcsft92b].

6.2.1.2. Carga do Arquivo de Recursos

As informações a respeito dos recursos definidos pelo programador no arquivo de recursos devem ser carregadas para a memória na fase de inicialização da aplicação. O arquivo de recursos é lido seqüencialmente e, à medida que os comandos de definição dos recursos (ícones, menus e quadros de diálogo) forem encontrados, suas informações serão carregadas para a tabela de janelas ou a tabela de menus da aplicação. Desta forma, quando for necessário criar qualquer um

desses recursos, durante a execução da aplicação, as informações necessárias estarão à disposição da aplicação.

6.2.2. Serviços do Sistema Operacional

De todos os serviços implementados pelo módulo do sistema operacional, o único serviço que apresenta características particulares quando implementado no ambiente Windows é a alocação de memória. Alguns detalhes sobre a implementação desse serviço serão apresentados a seguir.

6.2.2.1. Alocação de Memória

Apesar de várias opções de alocação de memória estarem disponíveis no Windows, o serviço de alocação de memória dinâmica da UNIFIC foi implementado de forma a alocar sempre memória global e fixa.

A área alocada é global para evitar problemas com o limite de memória disponível localmente para as aplicações, como foi explicado na seção 3.3.1.2.

A opção fixa foi selecionada por permitir que os blocos de memória alocados sejam acessados diretamente, sem que seja necessário exigir que o programador trave as áreas alocadas antes de acessá-las.

Como esse tipo de alocação pode causar problemas de gerenciamento de memória no ambiente Windows se várias aplicações UNIFIC alocarem grandes áreas de memória ao mesmo tempo, no futuro, será projetado um mecanismo que permita às aplicações UNIFIC beneficiarem-se do esquema de gerência de memória do Windows quando executadas neste ambiente.

6.2.3. Entrada de Dados Orientada a Eventos

No módulo de entrada de dados orientado a eventos, os aspectos de implementação mais importantes a serem considerados durante a implementação da biblioteca UNIFIC no ambiente Windows, são: conversão dos eventos, implementação do *loop* de eventos e recebimento dos eventos de *software*.

6.2.3.1. Conversão dos eventos

O sistema de recebimento de eventos original do Windows continua a ser utilizado pelo UNIFIC até que o evento gerado seja colocado na fila de eventos da aplicação. A partir deste momento a responsabilidade pela obtenção dos eventos e o seu envio para processamento passa ser do UNIFIC. Quando a UNIFIC obtém um evento da fila de eventos da aplicação, ela consulta as informações contidas no evento Windows e a partir destas informações transforma o evento em um evento UNIFIC.

6.2.3.2. Laço de Eventos

A obtenção dos eventos da fila de eventos da aplicação é realizada pelas aplicações UNIFIC através do seu laço de eventos. O laço de eventos de uma aplicação UNIFIC é responsável pela execução de várias ações, entre as quais pode-se citar, além da obtenção dos eventos da fila de eventos da aplicação, a sua interpretação e o seu envio para a função de processamento adequada. Para implementar esse laço como uma função única no ambiente Windows, as funções que realizam cada uma dessas tarefas na API Windows foram agrupadas, formando uma única função.

Existe, no entanto, uma condição particular que deve ser observada durante a implementação desse laço no ambiente Windows. Quando um quadro de diálogo sem modo

estiver aberto em uma aplicação, os eventos recebidos pelo sistema podem ser destinados ao quadro de diálogo ou a qualquer outra janela da aplicação (ou até mesmo do sistema). No ambiente Windows, sempre que esta situação ocorrs, o quadro de diálogo sem modo tem prioridade sobre o processamento do evento. Por isso, os eventos recebidos nessas circunstâncias são inicialmente enviados para serem processados pela função-ação do quadro de diálogo. O valor de retorno da função-ação indica se o evento foi realmente processado pelo quadro de diálogo. Caso não tenha sido, o evento será devolvido ao laço principal da aplicação que, então, o encaminhará para uma função-ação mais adequada.

Para garantir que as aplicações UNIFIC processem corretamente os eventos provenientes do teclado nessas circunstâncias, sempre que este tipo de evento for recebido pela aplicação torna-se necessário verificar se existe algum quadro de diálogo sem modo ativo na aplicação. Isto é feito percorrendo-se a tabela de janelas em busca de uma janela secundária ativa, que esteja sendo utilizada como um quadro de diálogo sem modo. Se este quadro de diálogo for encontrado, o evento é enviado inicialmente para a função-ação do quadro de diálogo sem modo. A partir do valor booleano retornado pela função-ação do quadro de diálogo, é possível verificar se o evento foi processado por esta função ou não. Caso o evento não tenha sido processado, ele é devolvido para o laço de eventos principal que, então, envia-o para o processamento convencional.

6.2.3.3. Recebimento dos Eventos de Software

Como explicado na seção 3.3.3.1, os eventos gerados no ambiente Windows a partir de *software* são enviados diretamente para a função-ação da janela a qual foram destinados. Para impedir que os eventos que se originam de *software* sejam enviados pelo Windows diretamente para as funções-ação das aplicações UNIFIC, a UNIFIC utiliza um mecanismo disponível no Windows, o *hook*.

O *hook* é um mecanismo que permite a interceptação dos eventos Windows antes que eles sejam recebidos pelas respectivas aplicações. Esse mecanismo permite que os eventos gerados no ambiente Windows sejam observados, modificados e até removidos. O *hook* utilizado pela UNIFIC para interceptar os eventos de *software* gerados no ambiente Windows é um *hook* oferecido pelo próprio sistema Windows. Nesse caso a UNIFIC utiliza o *hook* para verificar apenas se o evento foi enviado para a janela de alguma aplicação UNIFIC. Em caso afirmativo, o evento é convertido para um evento UNIFIC e só então é enviado para a função-ação da janela destino. Maiores informações a respeito da utilização de *hooks* no Windows podem ser encontradas nas referências [Mcsft90b] e [Mcsft92b].

6.2.4. Criação dos Objetos da Interface Gráfica do Usuário

Durante a implementação do módulo dos objetos da GUI da biblioteca UNIFIC no ambiente Windows, os pontos que mereceram maior destaque foram: registro das janelas, sistema de coordenadas e criação de menus.

6.2.4.1. Registro da Janela

A UNIFIC utiliza as informações fornecidas como parâmetro à função de criação de janelas para registrar a classe da janela no ambiente Windows. As informações utilizadas pela UNIFIC para o registro de uma classe são o nome do ícone, a cor de fundo e o nome da função-ação da janela. Depois que a classe da janela for registrada, a janela pode ser criada no ambiente Windows.

No caso dos quadros de diálogo, definidos a partir do arquivo de recursos, a UNIFIC utiliza uma classe pré-registrada que permite a criação deste tipo de janela. Essa classe especial associa corretamente o nome da função-ação, fornecida como parâmetro à função de criação do

quadro de diálogo, à janela utilizada pelo quadro de diálogo. Portanto, para se criar um quadro de diálogo no ambiente Windows, a UNIFIC obtém as informações correspondentes ao quadro de diálogo desejado da tabela de janelas e chama a função de criação de janelas disponível na API UNIFIC.

6.2.4.2. Sistema de Coordenadas

Um aspecto particular da criação de janelas, a ser considerado no ambiente Windows, é a relação entre as coordenadas fornecidas para uma janela e as características do dispositivo de vídeo utilizado. Se as coordenadas fornecidas pelo usuário, em *pixels*, forem aplicadas diretamente pela UNIFIC no ambiente Windows, é possível que as janelas apresentem tamanhos desproporcionais, de acordo com a resolução do dispositivo de vídeo utilizado. Para evitar grandes desproporções, antes de criar uma janela no ambiente Windows a biblioteca UNIFIC consulta o número de *pixels* por polegada oferecido pelo vídeo utilizado e ajusta, da melhor forma possível, a proporção dos objetos da GUI de acordo com a resolução do vídeo.

6.2.4.3. Criação de Menus

A criação dos menus UNIFIC no ambiente Windows é realizada de forma semelhante à criação dos quadros de diálogo. As informações sobre o menu que se deseja criar são obtidas da tabela de menus a partir do seu nome. Com estas informações a UNIFIC utiliza o conjunto de funções oferecidas pela API Windows para criar os menus e acrescentar os itens necessários, compondo o menu desejado.

O próximo capítulo fará uma avaliação dos objetivos deste trabalho e, em seguida, apresentará algumas conclusões e sugestões para trabalhos futuros.

Capítulo 7

7. Conclusão

Atualmente, um importante fator a ser considerado no desenvolvimento de aplicações de qualidade é a utilização de GUIs (Graphical User Interface). Basicamente, o desenvolvimento de aplicações com GUIs envolve dois aspectos importantes: o ponto de vista do programador e o ponto de vista do usuário. Do ponto de vista do programador, as GUIs oferecem toda a infra-estrutura necessária (como por exemplo as APIs - Application Programming Interface) para o desenvolvimento de aplicações com interfaces gráficas. Do ponto de vista do usuário, as aplicações com GUIs apresentam um *look-and-feel* consistente e agradável, o que torna mais fácil a interação entre o usuário e a aplicação. Devido a essas características, as GUIs estão sendo cada dia mais utilizadas na área de desenvolvimento de aplicações.

Um fator que inibe a utilização em larga escala das GUIs em diversas plataformas de *hardware*, na área de desenvolvimento de aplicações, é a falta de uma GUI padrão amplamente aceita pelo mercado.

Como visto no capítulo 1, a inexistência desse padrão gera problemas de portabilidade para as aplicações desenvolvidas para uma GUI que precisem ser transportadas para outros ambientes gráficos. Isso ocorre, porque a compatibilidade das aplicações, normalmente, restringe-se ao ambiente gráfico para o qual foram desenvolvidas. Enquanto não se estabelecer um padrão *de facto* para GUIs, uma forma de minimizar este problema é utilizar ferramentas, como por exemplo a UNIFIC, que facilitem o transporte das aplicações, entre diferentes GUIs.

Nos capítulos anteriores foram apresentados detalhes sobre o desenvolvimento de uma biblioteca de programação que unifica a programação entre diferentes GUIs. Os requisitos dessa biblioteca foram identificados e as características mais importantes das GUIs, disponíveis no mercado, foram estudadas. Além disso foi definido um modelo de programação unificado e implementada uma API baseada neste modelo.

A principal contribuição deste trabalho é a definição da API UNIFIC que oferece às aplicações, desenvolvidas a partir dela, portabilidade inerente no seu código fonte, para diversos ambientes gráficos.

A primeira seção deste capítulo avalia os objetivos propostos no início deste trabalho e a segunda apresenta as conclusões finais e algumas sugestões para trabalhos futuros.

7.1. Avaliação dos Objetivos do Trabalho

Os dois principais objetivos deste trabalho, definidos na seção 1.2, foram: 1) prover uma API baseada em um modelo de programação que unifique a programação entre diferentes GUIs; 2) mostrar a viabilidade desta API implementando-a em um ambiente gráfico. A seguir será feita uma análise crítica, verificando até que ponto estes objetivos foram atingidos.

1) Prover uma API baseada em um modelo de programação que unifique a programação entre diferentes GUIs.

A identificação dos requisitos básicos e específicos, apresentada no capítulo 2, estabelece as diretrizes para a definição da API de unificação. Enquanto os requisitos básicos garantem que as aplicações desenvolvidas com essa API podem ser facilmente implementadas e executadas em diferentes ambientes gráficos, os requisitos específicos garantem que a API oferece ao programador toda infra-estrutura para o desenvolvimento de aplicações com GUIs.

Através da análise dos modelos de programação adotados pelas GUIs Windows e Motif, feita no capítulo 3, foi possível obter subsídios para a definição do modelo de programação unificado. Apesar de apenas duas GUIs terem sido estudadas, praticamente todas as GUIs disponíveis no mercado possuem características que se enquadram em uma dessas duas GUIs.

A partir da identificação dos requisitos da API e do estudo do modelo de programção das GUIs selecionadas, apresentados nos capítulos 2 e 3, foi possível definir o **modelo de programação unificado** para a API UNIFIC. O modelo, apresentado no capítulo 4, define um conjunto de serviços que envolve operações com o sistema operacional, saída gráfica, troca de dados, entrada de dados orientada a eventos e objetos da GUI. O modelo de programação unificado pode ser implementado, diretamente, em diversos ambientes gráficos. Além disso, é importante observar que esse modelo serve como base para a inclusão de novas GUIs à API UNIFIC, mesmo que as novas GUIs não se enquadrem diretamente em nenhuma das filosofias de programação estudadas. Pois, parte substancial do tempo que seria gasto para se identificar as características mais importantes da nova GUI, do ponto de vista de unificação, é eliminado através deste estudo.

O tutorial, apresentado no capítulo 5, mostra como a API UNIFIC pode ser utilizada pelo programador para desenvolver as suas aplicações.

O estudo permitiu o desenvolvimento de uma API baseada em um modelo de programação que atende a todos os requisitos especificados no capítulo 2 e, por isto, unifica a programação entre diferentes GUIs. Entretanto, a fim de atender a esses requisitos de forma genérica, características interessantes, particulares a algumas GUI, tiveram que ser sacrificadas. Isso foi inevitável porque a adoção de características específicas a determinados ambientes tenderia a restringir a abrangência do modelo de programação unificado. Por exemplo, o modelo de

programação unificado não oferece ao programador fontes compatíveis com PostScript. Isso deve-se ao fato de que, em alguns ambientes gráficos, esse tipo de fonte ainda não está disponível. A inclusão desse tipo de fonte no modelo unificado exigiria um estudo específico mais profundo que estenderia por demais este trabalho. Um outro exemplo é a falta de um mecanismo que permita às aplicações UNIFIC trocarem dados entre si utilizando o esquema de envio de mensagens. Como este tipo de troca de dados pode ser inviável de implementar diretamente em vários ambientes gráficos, um estudo que propusesse a sua implementação de forma unificada também estaria fora do escopo deste trabalho.

Outro fator limitante relaciona-se ao *look-and-feel* das aplicações. Alguns objetos da GUI, simplesmente, não fazem parte do *look-and-feel* de outras. Esse é o caso dos menus de opções, presentes apenas no Motif, ou dos *combo boxes*, presentes apenas no Windows. Como este trabalho pretende manter o *look-and-feel* original do ambiente onde a aplicação está sendo executada, não teria sentido a UNIFIC oferecer objetos da GUI que não fossem familiares a todos os ambientes gráficos. A opção, nesse caso, foi restringir o universo de objetos da GUI para um conjunto mínimo aceitável.

Contudo, o modelo de programação unificado oferece ao programador toda infra-estrutura básica necessária ao desenvolvimento da maioria das classes de aplicações. No futuro, algumas modificações particulares deverão ser incorporadas à biblioteca, de forma a permitir que aplicações mais específicas também possam ser desenvolvidas a partir da API UNIFIC.

2) Mostrar a viabilidade da API através da sua implementação para um ambiente gráfico.

O ambiente gráfico selecionado para se implementar inicialmente a biblioteca UNIFIC foi o Windows. O capítulo 6 apresenta os detalhes da implementação dos principais módulos

componentes da API UNIFIC nesse ambiente. Esses módulos são: serviços do sistema operacional, saída gráfica, entrada de dados orientada a eventos e criação dos objetos da GUI. Como os módulos foram implementados sem problemas nesse ambiente e as aplicações desenvolvidas como teste para a API UNIFIC, também foram executadas com sucesso, nós acreditamos que o segundo objetivo deste trabalho tenha sido atingido.

Contudo, comparando-se o desempenho de uma aplicação UNIFIC com o desempenho de uma aplicação Windows, constata-se que a aplicação UNIFIC apresenta desempenho inferior durante a apresentação dos controles dos quadros de diálogo. Essa queda de desempenho deve-se à forma como os quadros de diálogo são criados internamente pela API UNIFIC. As informações a respeito de cada controle são lidas individualmente da tabela de janelas e os controles criados durante a apresentação do quadro de diálogo. Acreditamos que esse problema poderá ser bastante minimizado através da adoção de uma tabela *hash* para a manutenção das informações a respeito das janelas e com a otimização de alguns trechos de código específicos que também possam representar algum gargalo.

Apesar do objetivo formulado ter sido atendido, concluímos que a fim de comprovar, de fato, a viabilidade da biblioteca UNIFIC, seria necessário implementá-la em, pelo menos, mais um ambiente gráfico.

7.2. Conclusões Finais e Trabalhos Futuros

Acreditamos que o desenvolvimento uma biblioteca de programação para unificar, de forma plenamente satisfatória, todas as GUIs do mercado, exigiria um estudo mais abrangente. Esse estudo deveria tratar, profundamente, de aspectos como, por exemplo, internacionalização, fontes compatíveis com PostScript e mecanismos mais poderosos para o compartilhamento de

áreas de memória. Apesar deste trabalho não tratar especificamente de todos esses aspectos, as soluções aqui adotadas são genéricas o suficiente para atender aos requisitos básicos da maioria das aplicações e, dessa forma, constitui uma ferramenta válida para o desenvolvimento de aplicações portáveis entre diferentes GUIs.

Para tornar este trabalho mais completo, vários outros trabalhos poderiam ser desenvolvidos. Alguns dos trabalhos sugeridos são:

1) **Editor de recursos UNIFIC**: desenvolver um editor de recursos que ofereça facilidades para o programador UNIFIC definir os recursos da sua aplicação. Essa ferramenta deve permitir a definição de recursos por meio de uma interface gráfica e gerar, automaticamente, os arquivos .REC.

2) **Padrão Único de *Bitmaps***: definir um formato para *bitmaps* que permita ao programador utilizar uma única ferramenta e criar *bitmaps* compatíveis para todos os ambientes gráficos.

3) **Versão C++ da biblioteca UNIFIC**: implementar a versão C++ da API UNIFIC em todos os ambientes onde a UNIFIC estiver disponível.

4) **Definição da tabela de cores UNIFIC**: definir, para inclusão no módulo de saída gráfica, um método que permita ao próprio programador definir uma tabela contendo a escala de cores a ser utilizada por sua aplicação.

5) **Fontes Portáveis UNIFIC**: oferecer fontes compatíveis com PostScript para as aplicações UNIFIC.

6) **Processamento da tabela de teclas aceleradoras**: incluir, no modelo de entrada de dados da API UNIFIC, o processamento da tabela de teclas aceleradoras para os comandos da aplicação. A inclusão dessa tabela vai permitir que o programador ative os comandos da aplicação

através do teclado, dispensando o uso do apontador.

7) **Ferramenta de Criação de *Helps* UNIFIC**: desenvolver uma ferramenta que facilite o desenvolvimento de *helps* para as aplicações UNIFIC.

8) **Implementar a API UNIFIC em outros Ambientes Gráficos**: implementar a API UNIFIC em ambientes gráficos como, por exemplo, o Motif, Macintosh e Presentation Manager.

Bibliografia

Referências Bibliográficas

[Adler90] Adler, M. *Learning Windows Part II: Resources and the Menuing System*, Microsoft Systems Journal, Sep. 1990, pp. 75-87.

[Adler91a] Adler, M. *Learning Windows Part IV: Intergrating Controls and Dialog Boxes*, Microsoft Systems Journal, Jan. 1991, pp. 97-119.

[Adler91b] Adler, M. *Learning Windows Part V: Exploring the Graphics Device Interface*, Microsoft Systems Journal, Mar. 1991, pp. 75-88.

[Adler91c] Adler, M. *Learning Windows Part VI: Fonts, Bitmaps and Printing*, Microsoft Systems Journal, Jan. 1991, pp. 97-119.

[Berlage91] Berlage, T. *OSF/Motif: Concepts and Programming*, Addison Wesley, 1991.

[Côté92] Côté, R. G. *Code on the Move*, Byte, Vol. 17, N° 7, Jul. 1992, pp. 206-226.

[Dichter93] Dichter, C. *One For All*, UNIX Review, Vol. 11, N° 10, Oct. 1993, pp. 65-74.

[Dunphy91] Dunphy, E. *The UNIX Industry*, QED Technical Publishing Group, 1991.

[Gray91] Gray, P. A. *OpenSystems: A Business Strategy for the 1990's*, McGraw-Hill, 1991.

[Hayes89] Hayes, F. & Baran, N. *A Guide to GUIs*, Byte, Jul. 1989, pp. 250-257.

[Heller91] Heller, D. *Motif Programming Manual*, OReilly & Associates, Inc., 1991.

[Heller92] Heller, M. *Advanced Windows Programming*, John Wiley & Sons, Inc., 1992.

[Keller90] Keller, B. J. *A practical Guide to X Window Programming*, CRC Press, 1990.

[Kobara91] Kobara, S. *Visual Design with OSF/Motif*, Addison Wesley, 1991.

[Malamud92] Malamud, C. *Stacks: Interoperability in Today's Computer Networks*, Prentice Hall, 1992.

[Mcsft90a] Microsoft. *Windows Programmer's Reference*, Microsoft Press, 1990.

[Mcsft90b] Microsoft. *Windows Guide to Programming*, Microsoft Press, 1990.

[Mcsft92a] Microsoft, *TheWindows Interface: An Application Design Guide*, Microsoft Press, 1992.

[Mcsft92b] Microsoft, *Windows 3.1 - Programmer's Reference*, Microsoft Press, 1992.

[Myers89] Mayers, B. A. *Users-Interfaces Tools: Introduction and Survey*, IEEE Software, Jan. 1989, pp. 15-23.

[Norton90] Norton, P. & Yao P. *Windows 3.0: Power Programming Techniques*, Bantam Computer Books, 1990.

[Nye90] Nye, A. *Xlib Programming Manual*, O'Reilly Associates, Inc., 1990.

[Nye92] Nye, A. & O'Reilly, T. *X Toolkit Intrinsics Programming Manual*, O'Reil'ly & Associates, Inc., 1992.

[O'Reilly89] O'Reilly, T. *The Toolkits (and Politics) of X Windows*, UNIX World, Feb. 1989, pp. 66-72.

[OSF90] Open Software Foundation. *OSF/Motif: The Graphical User Interface for Open Systems*, A White Paper, OSF-1-WP4-1090-2, Oct. 1990, pp. 1-13.

[Petzold90] Petzold, C. *Programming Windows*, Microsoft Press, 1990.

[Petzold91] Petzold, C. *Windows 3.1 - Hello to TrueType, OLE, and Easier DDE; Farewell to Real Mode*, Sep. 91, pp. 17-26.

[Scheifler86] Sheifler, R. W. & Gettys, J. *The X Window System*, ACM Transactions on Graphics, Vol. 5, N° 2, Apr. 1986, pp. 79-109.

[Southerton90] Southerton, A. *Windows 3.0 Programming Primer*, Addison Wesley, 1990.

[Sun92] SunSoft, *Writing Applications for the Solaris Environment: A Guide for Windows Programmers*, Addison Wesley, 1992.

[Wilton91] Wilton, R. *Windows 3: Developer's Workshop*, Microsoft Press, 1991.

[XVT92a] XVT Software Inc., *GUI Portability Toolkit*, Jul. 1992, pp.1-6.

[XVT92b] XVT Software Inc., *XVT Design*, Jul. 1992, pp. 1-4.

[Yao91] Yao, P. *Windows 32-bit API Gives Developers Advanced Operating Systems Capabilities*, Microsoft Systems Journal, Nov/Dec. 1991, pp. 15-23.

[Yao92] Yao, P. *An Introduction to Windows NT Memory Management Fundamentals*, Microsoft Systems Journal, Jul/Aug. 1992, pp. 41-49.

Apêndice A

A1. Funções da Biblioteca UNIFIC

Neste apêndice serão apresentadas as funções oferecidas pela API UNIFIC. Para cada uma das funções existentes serão descritas sua sintaxe, funcionalidade, parâmetros de entrada e valores de retorno. As funções apresentadas a seguir classificam-se, basicamente, em: inicialização, serviços do sistema operacional, troca de dados entre aplicações, entrada de dados, saída gráfica e objetos da GUI.

A1.1. Inicialização

U_InicializaAplicação

Esta função realiza todas as operações de inicialização necessárias à execução de uma aplicação UNIFIC em qualquer ambiente gráfico. Ela deve ser a primeira função a ser chamada por uma aplicação UNIFIC.

Sintaxe **BOOL U_InicializaAplicação**(nomeRec)

Parâmetro	Tipo/Descrição
nomeRec	char * indica o endereço da cadeia de caracteres que contém o nome do arquivo de recursos da aplicação.

Retorno A função retornará TRUE se a inicialização foi realizada com sucesso e FALSE caso contrário.

A1.2. Serviços do Sistema Operacional

As funções oferecidas pela API UNIFIC para a realização dos serviços do sistema operacional subdividem-se em: gerenciamento de memória, manipulação de cadeias, entrada e saída em arquivos e execução de aplicações.

A1.2.1. Gerenciamento de memória

U_AlocaMemória

Esta função aloca uma área de memória cujo tamanho, em *bytes*, será especificado pelo parâmetro tamMem.

Sintaxe **U_HMEM U_AlocaMemória(** tamMem)

Parâmetro	Tipo/Descrição	
tamMem	int	indica o tamanho da área (em *bytes*) a ser alocada.

Retorno A função retornará o endereço de memória da área alocada ou NULL se houver algum erro e a área não puder ser alocada.

U_LiberaMemória

Esta função libera uma área de memória previamente alocada através da função **AlocaMemória.**

Sintaxe **U_HMEM U_LiberaMemória(** hMem)

Parâmetro	Tipo/Descrição
hMem	U_HMEM especifica endereço da área de memória a ser liberada.

Retorno A Função retornará NULL se a área for liberada com sucesso ou hMem caso contrário.

U_RealocaMemoria

Esta função aumenta ou diminui o tamanho de uma área de memória já alocada, sem que o seu conteúdo seja alterado.

Sintaxe **U_HMEN U_RealocaMemória(** hMem, tamMem)

Parâmetros	Tipo/Descrição
hMem	U_HMEM contém o endereço da área a ser realocada.
tamMem	int indica o novo tamanho da área de memória.

Retorno Esta função retornará o novo endereço da área de memória ou NULL se houver algum erro a área não puder ser realocada.

A1.2.2. Manipulação de Cadeias

U_ComparaCadeia

Esta função compara duas cadeias de caracteres e retorna a relação entre eles.

Sintaxe **int U_ComparaCadeia(** cadeia1, cadeia2)

Parâmetros	Tipo/Descrição
cadeia1	U_LSTRING contém o endereço da primeira cadeia a ser comparada.
cadeia2	U_LSTRING contém o endereço da segunda cadeia a ser comparada.

Retorno O valor retornado pela função indicará a relação entre as duas cadeias.

Valor	Significado
> 0	Indica que a primeira cadeia é menor do que a segunda.
= 0	Indica que as duas cadeias são iguais.
< 0	Indica que a primeira cadeia é maior do que a segunda.

U_ConcatenaCadeia

Esta função concatena a cadeia de caracteres cadeia2 no final da cadeia de caracteres cadeia1.

Sintaxe **U_LSTRING U_ConcatenaCadeia(** cadeia1, cadeia2)

Parâmetros	Tipo/Descrição
cadeia1	U_LSTRING contém o endereço da primeira cadeia a ser concatenada.
cadeia2	U_LSTRING contém o endereço da segunda cadeia a ser concatenada.

Retorno A função retornará o endereço da nova cadeia que contém as duas cadeias concatenadas.

U_CopiaCadeia

Esta função copia o conteúdo da segunda cadeia de caracteres para a primeira.

Sintaxe **BOOL U_CopiaCadeia(** cadeia1, cadeia2 **)**

Parâmetros	Tipo/Descrição
cadeia1	U_LSTRING contém o endereço da cadeia destino.
cadeia2	U_LSTRING contém o endereço da cadeia fonte.

Retorno A função retornará TRUE se a cadeia for copiada com sucesso e FALSE caso contrário.

U_ObtemTamanhoCadeia

Esta função fornece o número de caracteres contidos na cadeia especificada pelo parâmetro cadeia1.

Sintaxe **int U_ObtemTamanhoCadeia(** cadeia1 **)**

Parâmetros	Tipo/Descrição
cadeia1	U_LSTRING contém o endereço da cadeia a se obter o tamanho.

Retorno A função retornará o número de caracteres contidos na cadeia.

A1.2.3. Entrada e Saída em Arquivos

U_AbreArquivo

Esta função abre um arquivo já existente. Se um arquivo aberto para escrita não existir, ele será criado.

Sintaxe **U_HARQ U_AbreArquivo(** nomeArq, modAces **)**

Parâmetros	Tipo/Descrição	
nomeArq	U_LSTRING especifica o endereço da cadeia de caracteres que contém o nome do arquivo a ser aberto.	
modAces	U_JFLAG	indica o modo de abertura do arquivo.
	Valor	Significado
	A_LER	Abre um arquivo apenas para leitura.

A_ESCR	Abre um arquivo apenas para escrita.
A_LERESCR	Abre um arquivo para escrita e leitura.
A_EXCL	Indica exclusividade de acesso ao arquivo que será aberto.

Retorno Esta função retornará o descritor associado ao arquivo aberto ou -1 em caso de falha.

U_EscreveArquivo

Esta função permite que dados sejam escritos em um determinado arquivo. Para que os dados sejam escritos em um arquivo, é necessário que ele tenha sido aberto.

Sintaxe **int U_EscreveArquivo(** descArq, buffer, numBytes)

Parâmetros	Tipo/Descrição
descArq	U_HARQ especifica o descritor do arquivo no qual os dados serão escritos.
buffer	U_LSTRING especifica o endereço do *buffer* de caracteres que os dados a serem escritos no arquivo.
numBytes	int especifica o número de *bytes* a serem escritos no arquivo.

Retorno A função retornará o número de *bytes* realmente escritos no arquivo ou -1 em caso de falha.

U_FechaArquivo

Esta função fecha o arquivo especificado através descritor de arquivo descArq. Como resultado da execução dessa função, o arquivo não estará mais a disponível para leitura ou escrita de dados (a não ser que seja novamente aberto).

Sintaxe **int U_FechaArquivo(** descArq)

Parâmetros	Tipo/Descrição
descArq	especifica o descritor do arquivo a ser fechado.

Retorno A função retornará 0 se o arquivo for fechado com sucesso ou -1 em caso de falha.

U_LeArquivo

Esta função lê dados de um arquivo. Para que os dados possam ser lidos do arquivo, é necessário que ele tenha sido previamente aberto.

Sintaxe **int U_LeArquivo(** descArq, buffer, numBytes **)**

Parâmetros	Tipo/Descritor	
descArq	U_HARQ	indica o descritor do arquivo de onde os dados serão
buffer	U_LSTRING	especifica o endereço do *buffer* para onde os dados serão copiados.
numBytes	int	indica o número de *bytes* a serem lidos do arquivo.

Retorno O valor de retorno indicará o número de *bytes* efetivamente lidos do arquivo ou -1 em caso de falha. A função retornará EOF quando de encontrar fim de arquivo.

U_PosicionaArquivo

Esta função reposiciona o apontador de um arquivo, previamente aberto, para uma nova localização dentro do arquivo.

Sintaxe **int U_PosicionaArquivo(** descArq, offsetRolm, posOrig **)**

Parâmetros	Tipo/Descritor	
descArq	U_HARQ	especifica o descritor do arquivo cujo apontador será reposicionado.
offsetRolm	int	especifica o rolamento, em *bytes*, que o apontador deverá ser deslocado.
posOrig	U_JFLAG	especifica a posição a partir da qual o apontador será deslocado.

Valor	Significado
A_INIC	Desloca o apontador offsetDesl *bytes* a partir do início do arquivo.
A_ATU	Desloca o apontador offsetDesl *bytes* a partir da

posição em que se encontra o apontador.

A_FIM Desloca o apontador offsetDesl *bytes* a partir do fim do arquivo.

Retorno Retornará o valor, em *bytes*, do *offset* final do apontador em relação ao início do arquivo. A função retornará -1 em caso de falha.

A1.2.4. Execução de Aplicações

U_ExecAplic

Esta função carrega e executa uma aplicação UNIFIC. Se o diretório onde se encontra a aplicação a ser executada não for fornecido junto com o nome da aplicação, o arquivo será procurado inicialmente no diretório corrente e, em seguida, nos diretórios especificados pela variável de ambiente que contém os diretórios alternativos. Por exemplo, no sistema operacional DOS essa variável é a variável PATH.

Sintaxe LONG U_ExecAplic(nomeAplic, linCmd)

Parâmetros	Tipo/Descritor
nomeAplic	U_LSTRING especifica a cadeia de caracteres que contém o nome do arquivo a ser executado.
linCmd	U_LSTRING especifica a cadeia de caracteres que contém a linha de comando fornecida para a aplicação a ser executada.

Retorno A função retornará a instância do módulo carregado, se a função for executada com sucesso, ou um código de erro, caso contrário. Os seguintes códigos pode ser retornados:

Valor	Significado
0	Acesso fora da memória (*core dumped*).
1	Arquivo não encontrado.
2	Diretório não encontrado.
3	Arquivo inválido.

A1.3. Troca de Dados

U_AbreClipboard

Esta função abre o *clipboard* para que uma determinada aplicação possa acessá-lo. A abertura do *clipboard* garantirá que nenhuma outra aplicação terá permissão de modificar os dados contidos no *clipboard*.

Sintaxe **BOOL U_AbreClipboard(** idJan **)**

Parâmetros	Tipo/Descrição
idJan	U_IDJAN indentifica a janela a ser associada ao *clipboard* a ser aberto.

Retorno O retorno desta função indicará o estado do *clipboard*. O valor retornado será TRUE se o *clipboard* tiver sido aberto com sucesso e FALSE caso contrário.

U_ApagaDadosClipboard

Esta função remove os dados contidos no *clipboard* e libera a área de memória associada a estes dados.

Sintaxe **BOOL U_ApagaDadosClipboard()**

Esta função não tem parâmetros.

Retorno A função retornará TRUE se os dados forem removidos do *clipboard* com sucesso e FALSE caso contrário.

U_EscreveDadosClipboard

Esta função copia para o *clipboard* os dados contidos no endereço de memória especificado por hMem. A área de memória especificada por hMem será liberada após a cópia dos dados.

Sintaxe **U_HMEM U_EscreveDadosClipboard(** formDado, hMem **)**

Parâmetros	Tipo/Descrição
formDado	U_JFLAG especifica o formato dos dados que serão copiados para o

o *clipboard*.

Valor	Significado
CF_TEXTO	O dado possui formato texto, podendo conter caracteres de fim de linha, para indicar nova linha, e caracter NULL, para indicar o fim do texto.
CF_BITMAP	O dado fornecido é um apontador para um *bitmap*.

hMem U_HMEM especifica o endereço da área de memória que contém os a serem copiados para o *clipboard*.

Retorno A função retornará o endereço de memória que identifica os dados no *clipboard*. Este endereço será definido pelo próprio *clipboard*.

U_FechaClipboard

Esta função fecha o *clipboard*. A função **U_FechaClipboard** deve ser chamada quando a aplicação terminar de examinar ou modificar os dados contidos no *clipboard*. O fechamento do *clipboard* permitirá que outras aplicações possam acessá-lo.

Sintaxe **BOOL U_FechaClipboard()**

Esta função não tem parâmetros.

Retorno A função retornará TRUE se o *clipboard* for fechado e FALSE caso contrário.

U_LeDadosClipboard

Esta função lê dados do *clipboard*. Para que os dados possam ser lidos pela aplicação, o *clipboard* deve ter sido previamente aberto.

Sintaxe **int U_LeDadosClipboard(** formDado, hMem)

Parâmetros	Tipo/Descrição
formDado	U_JFLAG indica o formato dos dados a serem lidos do *clipboard*. Os de formatos que podem ser lidos do *clipboard* foram descritos na apresentação da função **U_EscreveDadosClipboard**, feita anteriomente.
hMem	U_HMEM indica o endereço de memória para onde os dados devem ser copiados.

Retorno A função retornará a quantidade de *bytes* lidos ou -1 em caso de falha.

U_VerificaFormatoClipboard

Esta função verifica se um determinado formato de dados está armazenado no *clipboard*.

Sintaxe **BOOL U_VerificaFormatoClipboard(** formDado **)**

Parâmetros	Tipo/Descrição
formDado	U_JFLAG indica o formato do dado a ser consultado. Os formatos de dados que podem ser encontrados no *clipboard* foram apresentados na descrição da função **U_EscreveClipboard**, feita anteriomente.

Retorno A função retornará TRUE se os dados armazenados no *clipboard* possuírem o formato desejado e FALSE caso contrário.

A1.4. Entrada de Dados

U_ConsultaFilaEvento

Esta função obtém o próximo evento da fila de eventos da aplicação destinado a janela idJan. O evento obtido não será removido da fila de eventos.

Sintaxe **BOOL U_ConsultaFilaEvento(** idJan, evento **)**

Parâmetro	Tipo/Descrição
idJan	U_IDJAN identificador da janela cujo evento será consultado.
evento	U_EVENTO endereço da estrutura de dados para onde as informações a respeito do evento consultado serão copiadas. O eventos podem ser do tipo:

Tipo evento	Significado
E_CRIA	Indica que uma janela será criada.
E_DESTROI	Indica que uma janela será destruída.
E_SAI	Indica o fim de execução de uma aplicação.
E_MOUSE_DUPLO	Indica que o usuário pressionou um dos botões do *mouse* duas vezes.
E_MOUSE_LIBERA	Indica que o usuário liberou um dos botões do *mouse*.

E_MOUSE_PRESS	Indica que o usuário pressionou um dos botões do *mouse*.
E_MOUSE_MOVE	Indica que o usuário moveu o dispositivo *mouse*.
E_CARAC	Indica que o usuário digitou um caractere através do teclado.
E_FOCO	Indica que uma janela deve receber o foco de entrada.
E_RESTAURA	Indica a existência de uma área na janela que precisa ser restaurada.
E_COMANDO	Indica que o usuário ativou um item de menu ou um controle da aplicação.

Retorno A função retornará TRUE se existir na fila algum evento para a janela especificada e FALSE caso contrário.

U_EnviaEventJan

Esta função envia uma mensagem diretamente para uma janela.

Sintaxe **LONG U_EnviaEventJan(** idJan, evento **)**

Parâmetros	Tipo/Descrição
idJan	U_IDJAN identificador da janela para a qual o evento será enviado.
evento	U_EVENTO especifica o tipo do evento a ser enviado.

Retorno A função retornará o mesmo valor de retorno da função ação da janela que processou o evento.

U_EnviaEventProc

Esta função envia o evento, contido na estrutura de dados evento, para ser processado pela função ação da janela a qual o evento pertence.

Sintaxe **BOOL U_EnviaEventProc(** evento **)**

Parâmetros	Tipo/Descrição
evento	U_EVENTO contém todas as informações que a aplicação necessita saber a respeito de um determinado evento. Os tipos de eventos que podem ser

enviados para processamento foram apresentados na descrição da função **U_ConsultaFilaEvento**.

Retorno A função retornará o mesmo valor retornado pela função ação que processou o evento.

U_FuncAcDef

Esta função oferece um processamento *default* para qualquer evento UNIFIC que não seja processsado por uma determinada aplicação. Todos os eventos que não forem explicitamente processados pela função ação associada à janela devem ser enviados para a função **U_FuncAcDef**.

Sintaxe **BOOL U_FuncAcDef(** evento **)**

Parâmetros	Tipo/Descrição
evento	U_EVENTO contém todas as informações que a aplicação necessita saber a respeito de um determinado evento. Os tipos de eventos que podem ser recebidos para processamento pela função **U_FuncAcDef** foram definidos na descrição da função **U_ConsultaFilaEvento**.

Retorno Nenhum.

U_LacoPrincipal

Esta função é um *loop* contínuo que lê os eventos da fila de eventos da aplicação, os interpreta e os envia para serem processados pela função ação associada a janela a qual o evento pertence. O *loop* se encerra quando lê o evento E_SAI da fila de eventos da aplicação.

Sintaxe **void U_LacoPrincipal(** evento **)**

Parâmetros	Tipo/Descrição
evento	U_EVENTO contém todas as informações que a aplicação necessita saber a respeito de um determinado evento. Os tipos de eventos que podem ser obtidos, pelo *loop* de eventos, da fila de eventos de uma aplicação UNIFIC foram definidos na descrição da função **U_ConsultaFilaEvento**.

Retorno Nenhum.

U_ObtemEvento

Esta função lê o próximo evento da fila de eventos da aplicação. Depois de lido, o evento é removido da fila de eventos.

Sintaxe **BOOL U_ObtemEvento(** evento)

Parâmetros	Tipo/Descrição
evento	U_EVENTO contém todas as informações que a aplicação necessita saber a respeito de um determinado evento. Os tipos de eventos que podem ser lidos da fila de eventos de uma aplicação UNIFIC foram definidos na descrição da função **U_ConsultaFilaEvento**.

Retorno A função retornará TRUE se o evento for lido com sucesso e FALSE caso contrário.

A1.5. Saída Gráfica

U_Arco

Esta função desenha um arco elíptico. O centro do arco é o centro de um retângulo que delimita a área do arco. O retângulo é especificado pelos parâmetros (x1, y1) e (x2, y2), e o arco inicia-se no ponto (x3, y3) e termina no ponto (x4, y4).

Sintaxe BOOL U_Arco(idJan, x1, y1, x2, y2, x3, y3, x4, y4)

Parâmetros	Tipo/Descrição
idJan	U_IDJAN indentificador da janela onde a linha será desenhada.
x1	int coordenada lógica x do canto superior esquerdo do retângulo que delimita a área do arco.
y1	int coordenada lógica y do canto superior esquerdo do retângulo que delimita a área do arco.
x2	int coordenada lógica x do canto inferior direito do retângulo que delimita a área do arco.
y2	int coordenada lógica y do canto inferior direito do retângulo que

delimita a área do arco.

x3	int	coordenada lógica x do ponto inicial do arco. O arco é desenhado no sentido anti-horário.
y3	int	coordenada lógica y do ponto inicial do arco.
x4	int	coordenada lógica x do ponto final do arco.
y4	int	coordenada lógica y do ponto final do arco.

Retorno A função retornará TRUE se o arco for desenhado com sucesso e FALSE caso contrário.

U_DefineAtribGrafic

Esta função define os objetos lógicos que devem ser utilizados por uma determinada janela durante a realização de operações gráficas. Os objetos lógicos definidos serão: lápis, pincel, fonte e cores. Se algum desses objetos não for explicitamente definido pelo programador, os atributos relativos a esse objeto não serão atualizados na janela.

Sintaxe **void U_DefineAtribGrafic(** idJan, atrib)

Parâmetros	Tipo/Descrição
idJan	U_IDJAN indentificador da janela a qual os atributos serão associados.
atrib	U_APATRIBDES endereço da estrutura de dados que contém os valores dos atributos a serem associados à janela para futuras operações gráficas.

Objeto	Tipo/Atributos	
LAPIS	short	larguraLapis
	U_JFLAG	estiloLapis
		Opções
		EL_SOLIDO
		EL_PONTIL
		EL_TRACEJ
		EL_NULO
	long	corLapis

Objeto	Tipo/Atributos	
PINCEL	U_JFLAG	padraoPincel
		Opções
		PP_HORIZ
		PP_VERTIC
		PP_DIAGESQ
		PP_DIAGDIR
		PP_DIAGCRUZ
	long	corPincel
FONTE	U_JFLAG	estiloFonte
		Opções
		EF_NORM
		EF_NEGRIT
		EF_SUBESCR
		EF_ITALIC
	U_JFLAG	familiaFonte
		Opções
		FF_SISTEM
		FF_COURIER
		FF_HELVETIC
	short	tamanhoFonte
COR	long	corFundo
	long	corFrente

Retorno Nenhum.

U_Elipse

Esta função desenha uma elipse. O centro da elipse é o centro do retângulo que delimita a área da elipse. O retângulo delimitador é especificado através dos parâmetros x1, y1, x2 e y2.

Sintaxe **BOOL U_Elipse(** idJan, x1, y1, x2, y2)

Parâmetro	Tipo/Descrição
idJan	U_IDJAN indentificador da janela onde a elipse será desenhada.
x1	int coordenada lógica x do canto superior esquerdo do retângulo que delimita a área da elipse.
y1	int coordenada lógica y do canto superior esquerdo do retângulo que delimita a área da elipse.
x2	int coordenada lógica x do canto inferior direito do retângulo que delimita a área da elipse.
y2	int coordenada lógica y do canto inferior direito do retângulo que delimita a área da elipse.

Retorno A função retornará TRUE se a elipse for desenhada com sucesso e FALSE caso contrário.

U_EscreveTexto

Esta função escreve uma cadeia de caracteres na área interna da janela especificada através do parâmetro idJan. A posição inicial do texto é definida pelos parâmetros x e y, fornecidos em relação à área interna da janela.

Sintaxe **BOOL U_EscreveTexto(** idJan, x, y, cadeia, tamCadeia)

Parâmetros	Tipo/Descrição
idJan	U_IDJAN indentificador da janela onde texto será escrito.
x	int especifica a coordenada lógica inicial x onde o texto será escrito.
y	int especifica a coordenada lógica inicial y onde o texto será escrito.
cadeia	U_LSTRING endereço da cadeia de caracteres a ser escrita.
tamBuffer	int especifica o número de caracteres contidos na cadeia.

Retorno A função retornará TRUE se a cadeia for escrita com sucesso e FALSE caso contrário.

U_EscreveTextoFormatado

Esta função escreve um texto formatado na área retangular especificada através do retângulo retText. O texto pode ser centralizado, justificado à esquerda ou à direita, dentro da área delimitada pelo retângulo.

Sintaxe **int U_EscreveTextoFormatado(** idJan, cadeia, tamBuffer, retText, formText **)**

Parâmetros	Tipo/Descrição
idJan	U_IDJAN indentificador da janela onde texto será escrito.
buffer	U_LSTRING endereço do *buffer* que contém os caracteres a serem escritos.
tamBuffer	especifica o número de caracteres contidos no *buffer*.
retText	U_APRET endereço da estrutura de dados RET que especifica a área retangular onde o texto deve será formatado.
formText	U_JFLAG especifica o tipo de formatação que deve ser aplicada ao texto.

Valor	Significado
TF_ESQ	Alinha o texto à esquerda da área retangular.
TF_CENTR	Centraliza o texto na área retangular.
TF_DIR	Alinha o texto à direita da área retangular.
TF_JUST	Justifica o texto dentro da área retangular.

Retorno A função retornará TRUE se o texto for escrito com sucesso e FALSE caso contrário.

U_Linha

Esta função desenha uma linha na área interna da janela especificada através do identificador idJan. As coordenadas fornecidas para o ponto inicial e o ponto final da linha devem ser especificadas em relação à área interna da janela.

Sintaxe **BOOL U_Linha(** idJan, x1, y1, x2, y2 **)**

Parâmetro	Tipo/Descrição
idJan	U_IDJAN indentificador da janela onde a linha será desenhada.

x1	int	coordenada lógica x do ponto inicial da linha.
y1	int	coordenada lógica y do ponto inicial da linha.
x2	int	coordenada lógica x do ponto final da linha.
y2	int	coordenada lógica y do ponto final da linha.

Retorno A função retornará TRUE se a linha for desenhada com sucesso e FALSE caso contrário.

U_Poligono

Esta função desenha um polígono que consiste de dois ou mais pontos (vértices), conectados por retas. O polígono é automaticamente fechado, se necessário, por meio de uma reta entre o último vértice e o primeiro.

Sintaxe **BOOL U_Poligono(** idJan, aPontos, nPontos)

Parâmetro	Tipo/Decrição
idJan	U_IDJAN indentificador da janela onde polígono será desenhado.
aPontos	U_APPONTOS especifica o endereço da lista que contém as coordenadas dos pontos que compõem o polígono.
nPontos	int indica o número de pontos do polígono.

Retorno A função retornará TRUE se o polígono for desenhado com sucesso e FALSE caso contrário.

U_Ponto

Esta função desenha um ponto na área interna da janela especificada através do parâmetro idjan. Os parâmetros de localização são especificados em relação a área interna da janela.

Sintaxe **BOOL U_Ponto(** idJan, x, y)

Parâmetro	Tipo/Descrição
idJan	U_IDJAN indentificador da janela onde o ponto será desenhado.
x	int indica a coordenada lógica x do ponto a ser desenhado.
y	int indica a coordenada lógica y do ponto a ser desenhado.

Retorno A função retornará TRUE se o ponto for desenhado com sucesso e FALSE caso

contrário.

U_ObtemAtribGrafic

Esta função obtem os objetos lógicos estão sendo utilizados por uma determinada janela durante a realização de operações gráficas. Os objetos lógicos obtidos serão: lápis, pincel, fonte e cores.

Sintaxe **void U_DefineAtribGrafic(** idJan, atrib **)**

Parâmetros	Tipo/Descrição
idJan	U_IDJAN indentificador da janela a qual se deseja obter os atributos.
atrib	U_APATRIBDES endereço do registro para onde os valores dos atributos de cada objeto lógico, utilizado pela janela, serão copiados.

Retorno Nenhum.

U_Retangulo

Esta função desenha um retângulo.

Sintaxe **BOOL U_Retangulo(** x1, y1, x2, y2 **)**

Parâmetro	Tipo/Descrição
idJan	U_IDJAN indentificador da janela onde retângulo será desenhado.
x1	int coordenada lógica x do canto superior esquerdo do retângulo.
y1	int coordenada lógica y do canto superior esquerdo do retângulo.
x2	int coordenada lógica x do canto inferior direito do retângulo.
y2	int coordenada lógica y do canto inferior direito do retângulo.

Retorno A função retornará TRUE se o retângulo for desenhado com sucesso e FALSE caso contrário.

A1.6. Objetos da Interface do Usuário

As funções disponíveis na biblioteca UNIFIC para a criação e gerência dos objetos da GUI classificam-se em: janelas, menus e quadros de diálogo.

A1.6.1. Janela

U_AtribuiJanFoco

Esta função atribui o foco de entrada à janela especificada através do identificador idJan.

Sintaxe **U_IDJAN U_AtribuiJanFoco(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN indentificador da janela a ser destruída.

Retorno A função retornará o identificador da janela que possuia, anteriormente, o foco de entrada. Esse valor será NULO se a janela não existir.

U_CapturaTeclado

Esta função faz com que toda entrada subseqüente, proveniente do *mouse*, seja enviada para a janela especificada pelo identificador idJan.

Sintaxe **U_IDJAN U_CapturaTeclado(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN indentificador da janela que deverá receber toda entrada do

Retorno A função retornará o identificador da janela que recebia, previamente, toda entrada do *mouse*. Este valor será NULO se a janela não existir.

U_CriaJanela

Esta função cria uma janela do tipo principal, secundária ou de controle. Através dessa função são especificados o título, estilo, posição inicial, tamanho, pai e menu da janela.

Sintaxe **U_IDJAN U_CriaJanela(** idPai, estJan, titJan, x, y, cx, cy, menJan, icoJan, funcJan **)**

Parâmetro	Tipo/Decrição
idPai	U_IDJAN especifica o identificador do pai da janela.
estJan	U_JFLAG especifica o estilo da janela que está sendo criada. Se a janela for do tipo principal ou secundária, os *flags* de tipo da borda, atributos da borda, estado da janela e tipo do quadro de diálogo podem ser combinados (tabela A.1). Se a janela for do tipo controle, os *flags* de estado da janela,

tipo de controle e atributos do controle podem ser combinados (tabela A.1). O programador ainda dispõe de um conjunto de *flags* específicos, (tabela A.2) definidos através da combinação de vários estilos básicos, que oferece uma forma rápida e prática de criar vários tipos de janelas.

titJan U_LSTRING especifica o endereço da cadeia de caracteres que contém o título da borda da janela.

x int para janelas principais ou secundárias, este parâmetro indica a coordenada x do canto superior esquerdo da borda da janela em relação à tela. Se este parâmetro for definido como CJ_DEFAULT, a UNIFIC cria a janela em uma posição *default*. Se a janela for uma janela de controle, x indica a posição inicial do controle em relação à área interna da janela pai. Nesse caso a constante CJ_DEFAULT não poderá ser utilizada.

y int para janelas principais ou secundárias, este parâmetro indica a coordenada y do canto superior esquerdo da borda da janela em relação à tela. Se este parâmetro for definido como CJ_DEFAULT, a UNIFIC cria a janela em uma posição *default*. Se a janela for uma janela de controle, y indica a posição inicial do controle em relação à área interna da janela pai. Nesse caso a constante CJ_DEFAULT não poderá ser utilizada.

cx int para janelas principais ou secundárias, este parâmetro indica a coordenada cx do canto inferior direito da borda da janela em relação à tela. tela. Se este parâmetro for definido como CJ_DEFAULT, a UNIFIC cria a a janela em uma posição *default*. Se a janela for uma janela de controle, cx indica a posição final do controle em relação à área interna da janela pai. Nesse caso a constante CJ_DEFAULT não poderá ser utilizada.

cy int para janelas principais ou secundárias este parâmetro indica a coordenada cy do canto inferior direito da borda da janela em relação à tela. Se este parâmetro for definido como CJ_DEFAULT, a UNIFIC cria a janela em uma posição *default*. Se a janela for uma janela de controle, cy

indica a posição final do controle em relação à área interna da janela pai. Nesse caso a constante CJ_DEFAULT não poderá ser utilizada.

menJan U_LSTRING endereço da cadeia de caracteres que contém o nome do menu associado à janela ou o código associado ao controle (depende do tipo da janela). Se a janela criada for uma janela principal ou secundária, o parâmetro menJan contém o nome do menu da janela. Esse parâmetro pode ser NULL se a janela não tiver menu. Se a janela criada for uma janela de controle, menJan indica o código associado ao controle.

icoJan U_LSTRING especifica o endereço da cadeia de caracteres que contém o nome do ícone associado à janela.

funcJan U_APPROC especifica o endereço da função ação associada à janela.

Retorno A função retornará o identificador da janela se a janela for criada com sucesso e FALSE caso contrário.

Tabela A.1	**Estilos básicos**
Tipo	**Significado**
J_PRINCIPAL	Cria uma janela principal (esta janela possui barra de título e borda).
J_SECUNDARIA	Cria uma janela secundária.
J_CONTROLE	Cria uma janela de controle como filha de uma janela principal ou secundária.
Tipo borda	**Significado**
JB_BORDA	Cria uma janela que possui borda simples.
JB_BORDAREDIM	Cria uma janela que possui borda redimensionável.
JB_BORDADUPL	Cria uma janela que possui borda dupla (esta janela não pode ter título).
Atributos borda	**Significado**
JA_VERBARROL	Cria uma janela que possui uma barra de rolamento horizontal.
JA_HORBARROL	Cria uma janela que possui uma barra de rolamento vertical.
JA_MENUSIST	Cria uma janela que apresenta um menu do sistema na sua borda.
JA_MAXICO	Cria uma janela que possui um ícone para a maximização do tamanho da janela.

Tabela A.1	**Estilos básicos**
JA_MINICO	Cria uma janela que possui um ícone para a minimização do tamanho da janela.
JA_TITULO	Cria uma janela que possui uma barra de título.
Estado	**Significado**
JE_VISIVEL	Cria uma janela que estará inicialmente visível ao usuário.
JE_INATIVA	Cria uma janela que estará inicialmente inativa.
JE_ICONICA	Cria uma janela que será apresentada, inicialmente, em forma de ícone.
JE_MINIMIZADA	Cria uma janela no seu tamanho mínimo.
JE_MAXIMIZADA	Cria uma janela no seu tamanho máximo.
Tipo Quadro	**Significado**
JD_MODO	Indica que a janela criada será um quadro de diálogo com modo.
JD_NAOMOD	Indica que a janela criada será um quadro de diálogo sem modo.
JD_SISMOD	Indica que a janela criada será um quadro de diálogo com modo no sistema.
Tipo Controle	**Significado**
JC_RADIO	Cria um controle do tipo botão de rádio.
JC_MARC	Cria um controle do tipo quadro de marcar.
JC_EDICAO	Cria um controle do tipo quadro de texto editável.
JC_ESTATICO	Cria um controle do tipo texto estático.
JC_GRUPO	Cria um controle do tipo grupo.
JC_LISTA	Cria um controle do tipo quadro de lista.
JC_COMAND	Cria um controle do tipo botão de comando.
Atributo controle	**Significado**
CC_DEFCOMAND	Indica que o botão de comando criado será o botão de comando *default* da janela.
CS_CENTR	Indica que o texto estático criado será centralizado.
CS_ESQ	Indica que o texto estático criado será alinhado à esquerda.
CS_DIR	Indica que o texto estático criado será alinhado à direita.
CL_MULTSEL	Indica que o quadro de lista criado pode ter mais de uma de suas opções

Tabela A.1	Estilos básicos
	selecionadas pelo usuário.
CL_ALFAB	Indica que as opções do quadro de diálogo serão ordenadas alfabeticamente.
CE_MULTILIN	Indica que o quadro de edição terá mais de uma linha para entrada de dados.
CE_MINUSC	Indica que os caracteres digitados na área editável do quadro de edição serão automaticamente convertidos para caracteres maiúsculos.
CE_MAIUSC	Indica que os caracteres digitados na área editável do quadro de edição serão automaticamente convertidos para caracteres minúsculos.
CG_FIMGRUPO	Indica o fim de um determinado grupo de controles.

Tabela A.1 Flags básicos disponíveis para a criação de janelas.

Tabela A.2	Estilos específicos
J_JANPRINC	Cria uma janela principal com as opções J_PRINCIPAL, JA_TITULO, JA_MENUSIST, JB_BORDAREDIM, JA_MINICO e JA_MAXICO.
J_JANSECUND	Cria uma janela secundária com as opções J_SECUNDARIA, JB_BORDA e JA_MENUSIST. Este estilo pode ser combinado com JA_TITULO.
JC_JANCOMAND	Cria uma janela de controle do tipo botão de comando com as opções J_CONTROLE e JC_COMAND. Esse tipo de estilo poder ser combinado com o *flag* JC_GRUPO.
JC_JANDEFCOMAND	Cria uma janela de controle do tipo comando com as opções J_CONTROLE, JC_COMAND e CC_DEFCOMAND. Esse este tipo de estilo pode ser combinado com a opção JC_GRUPO.
JC_JANRADIO	Cria uma janela de controle do tipo comando com as opções J_CONTROLE e JC_RADIO. Esse estilo pode ainda ser combinado com a opção JC_GRUPO.
JC_JANMARC	Cria uma janela de controle do tipo quadro de marcar com as opções J_CONTROLE e JC_CHECK. A opção JC_GRUPO pode ser acrescentada.
JC_JANESTATICO	Cria um controle do tipo texto estático com as opções J_CONTROLE,

Tabela A.2	Estilos específicos
	JC_ESTATICO e JS_ESQ. A opção JC_GRUPO pode ser acrescentada.
JC_JANLISTA	Cria um controle do tipo quadro de lista com as opções J_CONTROLE e JC_LISTA. As opções CL_MULTISEL e JC_GRUPO podem ser acrescentadas.
JC_JANEDICAO	Cria um controle do tipo quadro de edição com as opções J_CONTROLE, JC_EDICAO e JB_BORDA. A opção CE_MULTILIN, CE_MINUSC e CE_MAIUSC pode ser acrescentada a essa opção.

Tabela A.2 Flags pré-definidos para a criação de janelas com estilos específicos.

U_DestroiJanela

Esta função destroi a janela especificada através do identificador idJan. Além de fechar permanentemente a janela, as informações mantidas pela UNIFIC a respeito da janela, na tabela de janelas, são invalidadas, os menus associados à janela são destruídos e, se a janela possuir janelas filhas, elas também serão destruídas.

Sintaxe **BOOL U_DestroiJanela(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN especifica o indentificador da janela a ser destruída.

Retorno A função retornará TRUE se a janela for destruída com sucesso e FALSE caso

U_FechaJanela

Esta função minimiza uma janela. Se a janela especificada através do identificador idJan for uma janela principal ou secundária, a janela será transformada em um ícone.

Sintaxe **BOOL U_FechaJanela(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN indentificador da janela a ser fechada.

Retorno A função retornará TRUE se a janela for fechada com sucesso e FALSE caso contrário.

U_LiberaCaptura

Esta função libera a captura do *mouse* e volta a processar a entrada da forma convencional.

void U_CapturaTeclado()

Esta função não recebe parâmetros.

Retorno Nenhum.

U_LocalizaJanela

Esta função obtém a posição e o tamanho atual da janela especificada através do identificador idJan. As coordenadas da janela são copiadas para a estrutura RETANGULO. As dimensões da janela são fornecidas em relação ao vídeo como um todo e consideram, como parte da janela, a borda, as barras de rolamento e a barra de título, quando existentes.

Sintaxe **RETANGULO U_LocalizaJanela(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN indentificador da janela a qual se deseja obter a localização.

Retorno A função retornará a estrutura de dados RETANGULO que conterá as coordenadas do canto superior esquerdo e canto inferior direito da janela.

U_MostraJanela

Esta função torna a janela especificada através do identificador idJan visível para o usuário na tela do computador.

Sintaxe **BOOL U_MostraJanela(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN especifica o indentificador da janela a ser mostrada.

Retorno A função retornará TRUE se a janela for mostrada com sucesso e FALSE caso contrário.

U_MoveJanela

Esta função permite que a localização e o tamanho da janela, especificada através do identificador idJan, sejam alteradas. Os parâmetros de localização são especificados em *pixels*.

Sintaxe **BOOL U_MoveJanela(** idJan, x1, y1, x2, y2 **)**

Parâmetro	Tipo/Descrição
idJan	U_IDJAN especifica o indentificador da janela a ser movida.
x1	int indica a nova coordenada lógica x do canto superior esquerdo da janela.
y1	int indica a nova coordenada lógica y do canto superior esquerdo da janela.
x2	int indica a nova largura da janela.
y2	int indica a nova altura da janela.

Retorno A função retornará TRUE se a janela for movida com sucesso e FALSE caso contrário.

U_ObtemEstadoJanela

Esta função obtém o estado atual da janela especificada pelo identificador idJan. O operador "&" (AND binário) pode ser aplicado ao valor de retorno da função para testar existência dos *flags* JE_VISIVEL, JE_INATIVA, JE_MAXIMIZADA ou JE_ICONICA, e verificar o estado da janela.

Sintaxe **U_JFLAG U_ObtemEstadoJanela(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN indentificador da janela a qual se deseja obter o estado.

Retorno A função retornará uma combinação dos *flags* JE_VISIVEL, JE_INATIVA, JE_MAXIMIZADA e JE_ICONICA, cujo valor poderá ser testado através do operador para verificar quais dos *flags* estão presentes.

U_ObtemJanCaptura

Esta função retorna o identificador da janela que captura toda entrada proveniente do *mouse*.

Sintaxe **U_IDJAN U_ObtemJanCaptura()**

A função não recebe parâmetros.

Retorno A função retornará o identificador da janela que estiver recebendo toda entrada proveniente do *mouse*. Este valor será NULO se essa janela não existir.

U_ObtemJanelaAtiva

Esta função retorna o identificador da janela ativa. A janela ativa é a janela que possui o foco de entrada.

Sintaxe **U_IDJAN U_ObtemJanelaAtiva()**

Esta função não tem parâmetros.

Retorno A função retornará o identificador da janela ativa.

U_ObtemJanFoco

Esta função retorna o identificador da janela que possui o foco de entrada.

Sintaxe **U_IDJAN U_ObtemJanFoco()**

Esta função não tem parâmetros.

Retorno A função retornará o identificador da janela que possuir o foco de entrada.

U_ObtemPaiJanela

Esta função retorna o identificador do pai da janela especificada através do identificador idJan.

Sintaxe **U_IDJAN U_ObtemPaiJanela(** idJan **)**

Parâmetros	Tipo/Descrição
idjan	U_IDJAN indentificador da janela a qual se deseja obter o pai.

Retorno A função retornará o identificador da janela pai ou NULO se a janela não possuir pai.

A1.6.2. Menu

U_AtivaItem

Esta função torna ativo ou inativo o item do menu idMen cujo código associado é codItem.

Sintaxe **U_JFLAG U_AtivaItem(** idMenu, codItem, flagItem **)**

Parâmetros	Tipo/Descrição
idMenu	U_IDMENU identificador do menu cujo item se tornará ativo/inativo.
codItem	int especifica o código associado ao item que se tornará

flagItem	U_JFLAG	indica se o item se tornará ativo ou inativo.

Valor	Significado
MI_ATIVO	Indica que o menu será ativado.
MI_INATIVO	Indica que o item deve se tornar inativo.

Retorno A função retornará MI_ATIVO ou MI_INATIVO, de acordo com o estado anterior do item do menu.

U_CarregaMenu

Esta função cria um menu de acordo com as informações especificadas através do comando MENU no arquivo de recursos. O parâmetro nomeMenu indica o nome associado ao menu no arquivo de recursos. Depois de criado, menu será associado a janela cujo identificador é idJan.

Sintaxe U_IDMENU U_CarregaMenu(nomeMenu, idJan)

Parâmetro	Tipo/Decrição
nomeMenu	U_LSTRING especifica o endereço da cadeia de caracteres que contém o nome associado ao menu no arquivo de recursos.
idPai	U_IDJAN identificador da janela ao qual o menu estará associado.

Retorno A função retornará o identificador do menu criado ou NULO em caso de erro.

U_MarcaItem

Esta função marca ou desmarca um item do tipo opção de um menu. Se o *flag* especificado como o parâmetro flagItem for UM_MARCA, o item será marcado. Se o *flag* for UM_DESMARCA, o item será desmarcado.

Sintaxe U_JFLAG U_MarcaItem(idMenu, codItem, flagItem)

Parâmetros	Tipo/Descrição
idMenu	U_IDMENU identificador do menu cujo item será marcado/desmarcado.
codItem	int especifica o código associado ao item que será marcado/desmarcado.
flagItem	U_JFLAG indica se o item será marcado ou desmarcado.

Valor	Significado
MI_MARCA	Indica que o menu deve ser marcado.
MI_DESMARC	Indica que o item deve ser desmarcado.

Retorno A função retornará MI_MARCA ou MI_DESMARCA, de acordo com o estado anterior do item do menu.

U_ObtemEstadoMenu

A função retorna a combinação de *flags* que indica o estado atual do item de um menu. O *flag* retornado é uma combinação dos *flags* de atributo do item de um menu (MI_CHECA, MI_DESCHECA, MI_ATIVO, MI_INATIVO) com os *flags* de tipo (MI_ITEM, MI_SUBMENU, MI_SEPARADOR). Para verificar quais *flags* estão presentes no valor retornado pela função, o programador pode usar o operador "&" e testar a existência de cada *flag* individualmente.

Sintaxe **U_JFLAG U_ObtemEstadoItem(** idMen, codItem)

Parâmetros	Tipo/Descrição
idMen	U_IDMEN indentificador do menu que possui o item.
codItem	int código associado ao item o qual se deseja obter o estado.

Retorno A função retornará uma combinação dos *flags* MI_CHECA, MI_DESCHECA, MI_ITEM, MI_SUBMENU, MI_SEPARADOR. Para verificar quais *flags* estão presentes no valor retornado pela função, o operador '&' poderá ser utilizado.

U_ObtemMenu

Esta função fornece o identificador da barra de menu da janela especificada através do parâmetro idJan.

Sintaxe **U_IDMEN U_ObtemMenu(** idJan)

Parâmetros	Tipo/Descrição
idJan	U_IDJAN indentificador da janela a qual se deseja obter a barra de menu.

Retorno A função retornará o identificador da barra de menu da janela ou NULO se a janela não possuir barra de menu.

A1.6.3. Quadro de Diálogo

U_AtribuiIntDlgItem

Esta função converte o valor do inteiro, fornecido através do parâmetro numero, em uma cadeia, e o atribui à área editável do quadro de texto editável especificado através do código codCtrl.

Sintaxe **BOOL U_AtribuiIntDlgItem(** idDlg, codCtrl, numero, sinNum)

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle
codCtrl	int especifica o código associado ao controle durante a definição do quadro de diálogo no arquivo de recursos.
numero	int especifica o valor inteiro a ser atribuído à área editável do quadro de texto editável.
sinNum	BOOL indica o sinal atribuído à cadeia. O valor TRUE indica que o o número tem sinal.

Retorno A função retornará TRUE se o inteiro for associado com sucesso e FALSE caso contrário.

U_AtribuiCadeiaDlgItem

Esta função atribui a cadeia de caracteres, especificada através do parâmetro cadeia, à área editável de um quadro de texto editável ou título de outro tipo de controle.

Sintaxe **BOOL U_AtribuiCadeiaDlgItem(** idDlg, codCtrl, cadeia)

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle
codCtrl	int especifica o código associado ao controle, durante a definição do quadro de diálogo no arquivo de recursos.
cadeia	U_LSTRING especifica a cadeia de caracteres a ser atribuída à área editável do controle ou a seu título.

Retorno A função retornará TRUE se a cadeia for associada com sucesso e FALSE caso contrário.

U_CriaDialogo

Esta função cria um quadro de diálogo. As características desse quadro de diálogo devem ter sido definidas através do comando DIALOGO no arquivo de recursos da aplicação.

Sintaxe U_IDJAN U_CriaDialogo(nomeDlg, paiDlg, funcaoDlg)

Parâmetro	Tipo/Decrição
titJan	U_LSTRING especifica o endereço da cadeia de caracteres que contém o nome associado ao quadro de diálogo no arquivo de recursos.
idPai	U_IDJAN identificador da janela pai do quadro de diálogo.
funcJan	U_APPROC endereço da função ação a ser associada ao quadro de diálogo.

Retorno A função retornará o identificador do quadro de diálogo criado ou NULO se o quadro de diálogo não for criado.

U_CriaDlgMsg

Esta função é uma forma rápida que o programador UNIFIC dispõem para criar um quadro de mensagem.

Sintaxe U_JFLAG U_CriaQlgMsg(idJan, cadeiaMens, cadeiaTitulo, flagCmd)

Parâmetros	Tipo/Descrição
idJan	U_IDJAN identificador da janela pai do quadro de mensagem.
cadeiaMen	U_LSTRING especifica o endereço da cadeia de caracteres que contém a a mensagem a ser apresentada.
cadeiaTitulo	U_LSTRING especifica o endereço da cadeia de caracteres que contém o título a ser apresentado no quadro de mensagem.
flagCmd	U_JFLAG indica quais os tipos de botão de comando que devem estar presentes no quadro de mensagem a ser criado.

Valor	Significado
QM_OK	Indica que o quadro de mensagem terá apenas um botão de comando rotulado com OK.

QM_OKCANCEL	Indica que o quadro de mensagem terá um botão de comando com rótulo OK e outro com CANCEL.
QM_RETRYCANCEL	Indica que o quadro de mensagem terá um botão de comando com rotulo RETRY e outro com CANCEL.

Retorno A função retornará IDOK, IDCANCEL ou IDRETRY, de acordo com o botão de comando que for pressionado pelo usuário.

U_DestroiDialogo

Esta função libera os recursos e destroi a janela associada ao quadro de diálogo especificado através do parâmetro idDlg.

Sintaxe void U_DestroiDialogo(idDlg, retDlg)

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo a ser destruído.
retDlg	int especifica o valor a ser retornado pelo quadro de diálogo para a função **U_CriaDialogo**.

Retorno Nenhum.

U_EstaMarcadoRM

Esta função indica se um controle do tipo botão de rádio ou quadro de marcar está marcado, ou não.

Sintaxe BOOL U_EstaMarcadoRM(idDlg, cmdCtrl)

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle pertence.
cmdCtrl	int especifica o código associado ao controle durante a definição do quadro de diálogo no arquivo de recursos.

Retorno A função retornará TRUE se o controle estiver marcado e FALSE caso contrário.

U_MarcaBotaoMarc

Esta função marca ou desmarca um botão do tipo quadrode marcar, dependendo do *flag* fornecido como o parâmetro flagMarc.

Sintaxe **BOOL U_MarcaBotaoMarc(** idDlg, codMarc, flagMarc **)**

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN identificador do quadro de diálogo ao qual o botão de marcar pertence.
codMarc	int código associado ao botão que se deseja marcar.
flagMarc	BOOL indica se o botão será marcado ou desmarcado.

Valor	Significado
0	Indica que o botão será marcado.
1	Indica que o botão será desmarcado.

Retorno A função retornará TRUE se o botão estiver anteriormente marcado e FALSE caso contrário.

U_MarcaBotaoRadio

Esta função marca o controle do tipo botão de rádio, especificado através do comando idBotao, e desmarca todos os demais botões de rádio pertencentes ao mesmo grupo.

Sintaxe **BOOL U_MarcaBotaoRadio(** idDlg, codInic, codFim, codBotao **)**

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o botão pertence.
codInic	int código associado ao primeiro botão de rádio do grupo, durante a definição do quadro de diálogo no arquivo de recursos.
codFim	int código associado ao último botão de rádio do grupo, durante a definição do quadro de diálogo no arquivo de recursos.
codCtrl	int especifica o código, associado durante a definição do quadro de diálogo no arquivo de recursos, do botão de rádio a ser marcado.

Retorno A função retornará TRUE se o botão de rádio associado ao código cmdCtrl já estiver marcado e FALSE caso contrário.

U_MostraLista

Esta função preenche o quadro de lista, identificado através do código codCtrl, com as opções contidas na cadeia de caracteres listOpc.

Sintaxe **BOOL U_MostraLista(** idDlg, codCtrl, escDef, listOpc)

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle
codCtrl	int especifica o código associado ao quadro de lista durante a definição do quadro de diálogo no arquivo de recursos.
escDef	U_LSTRING especifica o endereço da cadeia que possui a opção *default* da lista.
listOpc	U_LSTRING * especifica o endereço da *lista* de cadeias que contém todas as opções do quadro de lista..

Retorno A função retornará TRUE se as opções da lista forem apresentadas com sucesso e FALSE caso contrário.

U_MostraListaDir

Esta função inicializa um controle do tipo quadro de lista com o nome dos arquivos contido no diretório especificado através do parâmetro lpDir.

Sintaxe **BOOL U_MostraListaDir(** idDlg, lpDir, codList, codEstDir, tipoArq)

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle pertence.
lpDir	U_LSTRING especifica o endereço da cadeia que possui o nome diretório a ser pesquisado.
codList	int especifica o código associado quadro de lista durante a definição do quadro de diálogo no arquivo de recursos.

codEstDir	int especifica o código associado ao texto estático que representa o nome do diretório cujos arquivos estão sendo pesquisados.	
tipoArq	U_JFLAG especifica o tipo de arquivo a ser apresentado na lista.	
	Valor	Significado
	LD_ARQ	Arquivo.
	LD_SUBDIR	Subdiretório.
	LD_ARQLER	Arquivos apenas de leitura.
	LD_ARQLERESC	Arquivos de leitura e escrita.

Retorno A função retorna TRUE se os arquivos foram mostrados com sucesso e FALSE caso contrário.

U_ObtemEstadoCtrl

Esta função obtém o estado de uma janela do tipo controle.

Sintaxe **U_JFLAG U_ObtemEstadoCtrl(** idDlg, codCtrl **)**

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle
codCtrl	int especifica o código associado ao controle durante a definição do quadro de diálogo no arquivo de recursos.

Retorno A função retornará uma combinação dos atributos JE_VISIVEL e JE_INATIVA. Para quais dessas constantes estão presentes no estado da janela, o operador '&' pode ser usado.

U_ObtemIdControle

Esta função retorna o identificador da janela de controle cujo código associado é cmdDlg no arquivo de recursos.

Sintaxe **U_IDJAN U_ObtemIdCtrl(** idDlg, codCtrl **)**

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle
codCtrl	int especifica o código associado ao controle, durante a definição do

quadro de diálogo no arquivo de recursos.

Retorno A função retornará o identificador da janela de controle ou NULO se algum dos parâmetros fornecidos forem inválidos.

U_ObtemIntDlgItem

Esta função transforma o texto associado ao quadro de texto editável, identificado pelo código codCtrl, em um valor inteiro. Se o parâmetro sinNum for diferente de zero, o caractere que representa o sinal de menos (-), no início da cadeia, será transformado em um sinal negativo.

Sintaxe **BOOL U_ObtemIntDlgItem(** idDlg, codCtrl, numero, sinal **)**

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle
codCtrl	int especifica o código associado ao controle, durante a definição do quadro de diálogo no arquivo de recursos.
numero	int conterá o inteiro obtido do quadro de texto editável.
sinal	BOOL indica se o número a ser obtido tem sinal. Positivo para TRUE e negativo para FALSE.

Retorno A função retornará TRUE se o inteiro for obtido com sucesso e FALSE caso contrário.

U_ObtemOpcaoLista

Esta função obtém a opção selecionada pelo usuário em um quadro de lista.

Sintaxe **BOOL U_ObtemOpcaoLista(** idDlg, codList, cadeia **)**

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o quadro de lista pertence.
codCtrl	int especifica o código associado ao quadro de lista, durante a definição do quadro de diálogo no arquivo de recursos.
buffer	U_LSTRING especifica o endereço do *buffer* que receberá a opção selecionada.

Retorno A função retornará TRUE se a opção selecionada for obtida com sucesso e FALSE caso contrário.

U_ObtemCadeiaDlgItem

Esta função obtém a cadeia associada à área editável do quadro de controle editável ou título do controle, cujo código de identificação é codCtrl.

Sintaxe **BOOL U_ObtemCadeiaDlgItem(** idDlg, codCtrl, cadeia, tamCadeia)

Parâmetros	Tipo/Descrição
idDlg	U_IDJAN indentificador do quadro de diálogo ao qual o controle pertence.
codCtrl	int especifica o código associado ao controle durante a definição do quadro de diálogo no arquivo de recursos.
cadeia	U_LSTRING especifica o endereço da cadeia que receberá o conteúdo da área editável ou título do controle.
tamCadeia	int especifica o tamanho máximo da cadeia a ser copiada.

Retorno A função retornará TRUE se a cadeia for obtida com sucesso e FALSE caso contrário.

Apêndice B

B1. Comandos do Arquivo de Recursos

Este apêndice descreve os comandos disponíveis na API UNIFIC para a definição do conteúdo do arquivo de recursos da aplicação. Os comandos a serem discutidos serão: ICONE, MENU e DIALOGO.

Na notação utilizada para a definição dos comandos, as palavras escritas com letras maiúsculas e em negrito (ex.: **ICONE**) indicam a palavra-chave do comando. As palavras iniciadas com letras minúsculas e em negrito (ex.: **nomeID**) indicam os parâmetros do comando. Os colchetes serão utilizados para indicar parâmetros ou comando opcionais (ex.: **[atributos]**).

B1.1. Comando Icone

O comando ICONE indica a definição de um ícone no arquivo de recursos da aplicação. Esse comando associa um identificador ao arquivo *bitmap* que contém a imagem gráfica através da qual o ícone será representado. Quando a aplicação necessitar acessar o referido ícone, o identificador associado ao ícone, através do comando ICONE, será utilizado.

O comando ICONE tem o seguinte formato:

ICONE nomeID nomeArquivo

- **nomeID**: este parâmetro especifica o identificador único, número ou alfabético, que

será utilizado pela aplicação para referenciar o ícone.

• **nomeArquivo**: este parâmetro é uma cadeia de caracteres ASCII que especifica o nome do arquivo *bitmap* associado ao ícone. O nome do diretório do arquivo deve preceder o nome do arquivo *bitmap*, caso o arquivo não se encontre no diretório de trabalho da aplicação.

B1.2. Comando Menu

O comando MENU permite que o programador defina um menu no arquivo de recursos da aplicação. Através desse comando o programador especificará várias informações que definirão a aparência e funcionalidade do menu. Esse comando será usado, basicamente, na primeira versão UNIFIC para definição da barra de menu das janelas.

O comando MENU tem o seguinte formato genérico:

```
MENU nomeMenu
{
        comando ITEM
                ou
        comando SUBMENU
}
```

• **nomeMenu** especifica o nome ou número que será usado pela aplicação para identificar o menu.

No corpo do comando MENU podem ser utilizados comandos do tipo ITEM ou SUBMENU. Esses comandos podem ser encontrados em qualquer número e a ordem em que forem definidos indicará a ordem dos itens do menu. A funcionalidade e o formato desses dois comando serão discutidos a seguir:

B1.2.1. Item

O **comando ITEM** será utilizado para definir os itens da barra de menu que devam realizar uma ação imediata. O comando ITEM apresenta o seguinte formato genérico:

ITEM titulo codigo [atributos]

• **titulo**: este parâmetro é uma cadeia de caracteres ASCII, inserida entre aspas duplas, que especifica o título do item que está sendo definido. O caractere que representará a *hotkey* desse item deve ser precedido pelo caractere padrão '&'. Se for desejo do programador utilizar o caractere '&' como parte do título do item, é necessário que dois *ampersans* sejam incluídos na cadeia (&&).

• **codigo**: este parâmetro é um inteiro único que especifica o código a ser gerado pela aplicação quando o usuário selecionar o item do menu. Esse código será enviado para a janela que possuir o menu quando esse tipo de seleção ocorrer.

• **atributos**: este parâmetro especifica um ou mais opções que modificarão a aparência e/ou o comportamento do item do menu. As opções disponíveis para os itens definidos no primeiro nível do comando MENU estão definidos na tabela B.1.

Tabela B.1

Opção	Descrição
MI_ATIVO	Indica que o item do menu poderá ser selecionado pelo usuário. Esta é a opção *default* do item.
MI_INATIVO	Indica que o item do menu não poderá ser selecionada pelo usuário.

Tabela B.1 Atributos disponíveis para o comando ITEM no menu principal.

B1.2.2. Submenu

O **comando SUBMENU** indica o início da definição de um submenu. O submenu (definido através de um menu *dropdown*), é um item de menu especial, que apresentará para o usuário uma lista de itens de menu quando for selecionado. O comando SUBMENU apresenta o seguinte formato genérico:

SUBMENU nomeSubmenu codigo [atributos]
{
comando ITEM
ou
comando SEPARADOR
ou
comando SUBMENU
}

• **nomeSubmenu**: este parâmetro é uma cadeia de caracteres ASCII, inserida entre aspas duplas, que especifica o nome do título associado ao submenu.

• **codigo**: este parâmetro é um inteiro único que especifica o código a ser gerado pela aplicação quando o usuário selecionar o submenu. Esse código será enviado para a janela que possuir o submenu quando esse tipo de seleção ocorrer.

• **atributos**: este parâmetro especifica um ou mais opções que modificarão a aparência e/ou o comportamento do submenu. As opções disponíveis para um submenu são as mesmas apresentadas anteriormente para o comando ITEM.

No corpo do comando SUBMENU podem ser encontrados comandos do tipo ITEM, SEPARADOR ou até mesmo outros comandos do tipo SUBMENU. Esses comandos serão explicados a seguir.

B1.2.2.1. ITEM no submenu

O **comando ITEM** é o mesmo já definido anteriormente, mas existem alguns atributos adicionais específicos para os itens de um submenu. Os atributos adicionais para esse tipo de item estão especificados na tabela B.2. Esse parâmetro pode ser definido como a combinação de um ou mais dessas opções, separadas por vírgulas ou espaços em branco.

Tabela B.2

Opção	Descrição
MI_OPCAO	Indica que o item definido será um item do tipo opção, mas este item não se apresentará automaticamente marcado.
MI_MARCA	Indica que o item definido será um item do tipo opção e estará marcado.
MI_DESMARCA	Indica que o item de menu do tipo opção será desmarcado.

Tabela B.2 Atributos disponíveis para o comando ITEM em um submenu.

B1.2.2.2. Separador

O **comando SEPARADOR** é apenas uma opção estética disponível para separar com uma linha um determinado grupo de itens em um menu *dropdown* ou cascata. O comando SEPARADOR apresenta o seguinte formato genérico:

SEPARADOR

Apenas a palavra-chave SEPARADOR será necessária para a definição desse tipo de item.

B1.2.2.3. SUBMENU no submenu

O comando SUBMENU, nesse caso, será utilizado para definir os menus em cascata ativados através dos itens do submenu.

B1.3. Comando Quadro de Diálogo

O comando DIALOGO é o comando padrão da UNIFIC para a definição dos quadros de diálogo da aplicação. O comando DIALOGO apresenta o seguinte formato genérico:

```
DIALOGO idDlg  x,  y,  cx,  cy
[ESTILO opcoes]
[TITULO "titulo"]
{
        tipoControle  idComando  x, y, cx, cy,  "tituloControle"  [modificador]
                      .
                      .
                      .
}
```

• **idDlg**: este parâmetro é um valor inteiro ou uma cadeia de caracteres ASCII, única, que será utilizada pela aplicação como identificador do quadro de diálogo.

• **x**, **y**, **cx**, **cy**: estes parâmetros especificam, respectivamente, a coordenada do canto superior esquerdo e a coordenada do canto inferior direito do quadro de diálogo. Todas as medidas devem ser fornecidas em *pixels*.

No corpo do comando DIALOGO podem ser usados os subcomandos opcionais ESTILO, TITULO e os subcomandos tipoControle. Esses comando serão explicados a seguir.

B1.3.1. ESTILO

O subcomando **ESTILO** é opcional e permite que o programador defina o tipo e comportamento da janela do quadro de diálogo. O parâmetro **opcoes** desse comando poderá ser definido como uma combinação, através do operador '|' (OU), de uma ou mais opções apresentadas na tabela B.3.

Tabela B.3

Opção	Descrição
JB_BORDADUPL	Indica que o quadro de diálogo criado terá borda dupla. Esse tipo de borda não permite a existência da barra de título na janela.
JB_BORDA	Indica que o quadro de diálogo criado terá borda simples.
JA_TITULO	Indica que o quadro de diálogo apresentará barra de título.
JA_MENUSIST	Indica que o quadro de diálogo apresentará um menu do sistema em sua borda.
JD_MODO	Indica que o quadro de diálogoapresentará o comportamento de um quadro de com modo.
JD_NAOMOD	Indica que o quadro de diálogo apresentará o comportamento de um quadro de diálogo sem modo.
JD_SISMOD	Indica que o quadro de diálogo apresentará o comportamento de um quadro de diálogo com modo no sistema.

Tabela B.3 Atributos disponíveis para o subcomando ESTILO.

B1.3.2. TITULO

O subcomando **TITULO** será utilizado pelo programador para definir o título do quadro de diálogo. O parâmetro **titulo** desse comando é uma cadeia de caracteres ASCII, inserida entre aspas duplas, que especificará o título a ser apresentado na barra de título da janela associada ao quadro de diálogo. Se o comando TITULO não for definido para um quadro de diálogo, a barra de título, quando existente, permanecerá vazia.

B1.3.3. tipoControle

O subcomando **tipoControle** será utlilizado para a criação de qualquer tipo de controle. De acordo com a palavra-chave especificada como **tipoControle**, as opções que estão disponíveis

como **modificadores** do comando, variam. A tabela B.4 apresenta as palavras-chaves que podem ser utilizadas como o primeiro parâmetro do comando **tipoControle** e seus respectivos modificadores.

• **idComando**: este parâmetro é um inteiro único que especifica o código a ser enviado para a janela que contém o controle, quando este controle for selecionado pelo usuário.

• **x, y, cx, cy**: estes parametros especificam, respectivamente, a coordenada do canto superior esquerda e a coordenada do canto inferior direito do controle em relação à área interna da janela pai do controle. Essas coordenadas são especificadas em *pixels*.

• **tituloComando**: estes parâmetro é uma cadeia de caracteres ASCII, incluído entre aspas duplas, que especifica o título do controle que está sendo definido. É uma cadeia de caracteres ASCII, incuída entra aspas dupla, que especifica o título do controle que está sendo definido.

Tabela B.4

Tipo controle	Modificadores
JC_COMAND	JE_INATIVA - indica que o botão de comando não poderá ser inicialmente selecionado pelo usuário. CC_DEFCOMAND - indica que o botão de comando definido será o botão de comando *default* do quadro de diálogo. JC_GRUPO - indica que o botão de comando será o primeiro controle de um determinado grupo. CG_FIMGRUPO - indica que o controle será o último controle do grupo que está sendo definido.
JC_MARC	JE_INATIVA - indica que o quadro de marcar não poderá ser inicialmente selecionado pelo usuário. JC_GRUPO - indica que o quadro de marcar será o primeiro controle de um determinado grupo.

Tabela B.4

Tipo controle	Modificadores
	CG_FIMGRUPO - indica que o controle será o último controle do grupo que está sendo definido.
JC_RADIO	JE_INATIVA - indica que o botão de rádio não poderá ser inicialmente selecionado pelo usuário.
	JC_GRUPO - indica que o botão de rádio será o primeiro controle de um determinado grupo.
	CG_FIMGRUPO - indica que o controle será o último controle do grupo que está sendo definido.
JC_ESTATICO	JE_INATIVA - indica que o texto estático não poderá ser inicialmente selecionado pelo usuário.
	CS_CENTR - indica que o texto estático será centralizado.
	CS_ESQ - indica que o texto estará alinhado à esquerda.
	CS_DIR - indica que o texto estático estará alinhado à direita.
	JC_GRUPO - indica que o texto estático será o primeiro controle de um determinado grupo.
	CG_FIMGRUPO - indica que o controle será o último controle do grupo que está sendo definido.
JC_EDICAO	JE_INATIVA - indica que quadro de texto editável não poderá ser inicialmente selecionado pelo usuário.
	CE_MINUSC - indica que todos os caracteres digitados pelo usuário, na área editável quadro de texto editável, serão automaticamente convertidos para caracteres
	CE_MAIUSC - indica que todos os caracteres digitados pelo usuário, na área editável quadro de texto editável, serão automaticamente convertidos para caracteres
	CE_MULTILIN - indica que a área editável do quadro de texto editável terá mais de uma linha para a entrada de dados.
	JB_BORDA - indica que a área editável do quadro de texto editável apresentará uma

Tabela B.4

Tipo controle	Modificadores
	borda. JC_GRUPO - indica que o quadro de texto editável será o primeiro controle de um determinado grupo. CG_FIMGRUPO - indica que o controle será o último controle do grupo que está sendo definido.
JC_LISTA	JE_INATIVA - indica que quadro de lista não poderá ser inicialmente selecionado pelo o usuário. CL_ALFAB - indica que o quadro de lista presentará sua opções automaticamente ordenadas alfabeticamente. CL_MULTSEL - indica que mais de uma opção poderá ser selecionada pelo usuário no quadro de lista. JC_GRUPO - indica que o quadro de lista será o primeiro controle de um determinado grupo. CG_FIMGRUPO - indica que o controle será o último controle do grupo que está sendo definido.

Tabela B.4 Opções disponíveis para o subcomando tipoControle e seus respectivos modificadores.

Lista de Abreviaturas

ABI *Application Binary Interface*. A abreviatura ABI refere-se a uma interface de aplicação que ofereçe portabilidade binária entre máquinas distintas.

API *Application Programming Interface*. A abreviatura API designa uma biblioteca de funções padronizadas, colocada à disposição do programador para permitir o desenvolvimento de aplicações com um alto nível de abstração, facilitando o trabalho de programação.

DOS DOS é a abreviatura utilizada para designar o sistema operacional MS-DOS desenvolvido pela Microsoft. O MS-DOS é um sistema operacional desenvolvido para ambiente monotarefa e monousuário.

GDI *Graphic Device Interface*. GDI é a abreviatura usada para designar a biblioteca de funções gráficas, independente de dispositivos, da API Windows.

GUI *Graphical User Interface*. A abreviatura GUI refere-se a um tipo de interface que utiliza símbolos gráficos para representar objetos em vídeos mapeados a *bits*. Cada objeto da GUI possui aparência e comportamento (*look-and-feel*) uniformes, de acordo com o grupo ao qual pertence. Janelas, menus e ícones são exemplos de alguns desses objetos.

ISO *International Standards Organization*. A abreviatura ISO refere-se a uma associação internacional de países membros, cada um dos quais representado por sua organização de estabelecimento de padrões. A ISO estabelece padrões gerais para comunicações e troca de informações.

LAN *Local Area Network*. LAN é a abreviatura usada para designar um grupo de computadores e outros dispositivos, dispersos por uma área relativamente limitada (um departamento ou prédio comercial, por exemplo), conectados entre si através de um canal de comunicação. O canal de comunicação permite que os dispositivos interajam uns com os outros.

MMU *Memory Management Unit*. A abreviatura MMU refere-se a uma unidade de hardware responsável pelo policiamento de endereços lógicos utilizados por uma aplicação, e o mapeamento destes para os endereços físicos na memória.

NT *New Technology*. NT é a abreviatura utilizada para designar o sistema operacional Windows NT desenvolvido pela Microsoft. O Windows NT é um sistema operacional projetado para ambientes de redes heterogêneas.

OSF *Open Software Foundation*. A abreviatura OSF é utilizada para designar uma organização formada por empresas como a Hewlett-Packard, Digital, IBM, e dezenas de outras corporações, que tem a finalidade de desenvolver especificações de *software* e tecnologia para sistemas abertos.

PC *Personal Computer*. A abreviatura PC refere-se a um computador *desktop* desenvolvido pela IBM ou um clone fabricado por terceiros.

RGB *Red, Green, Blue*. A abreviatura RGB refere-se a um sistema de geração de cores onde as cores são definidas através da especificação da intensidade de três cores primárias: vermelho, verde e azul. Como um *byte* é reservado para a especificação de cada uma dessas cores, esse sistema oferece cerca de 16 milhões de combinações únicas para cores.

Win32 Win32 refere-se a uma extensão da API Windows projetada para 32 *bits*. Essa extensão faz com que o Windows dê suporte a características avançadas como: suporte para múltiplos processadores, processamento distribuído, ambiente de rede e segurança.