UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO CANÔNICO DE FERRAMENTA PARA DESENVOLVIMENTO DE INTERFACE COM O USUÁRIO

por

MARCELO SOARES PIMENTA

Dissertação submetida como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Carlos Alberto Heuser

Orientador

Porto Alegre, janeiro de 1991

CATALOGAÇÃO NA PUBLICAÇÃO

Pimenta, Marcelo Soares

Um modelo canônico de ferramenta para desenvolvimento de interface com o usuário. Marcelo Soares Pimenta. - Porto Alegre : CPGCC da UFRGS, 1991

246 p.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1991.

Orientador : Heuser, Carlos Alberto.

Dissertação: Interface com o Usuário, Ferramentas para Desenvolvimento de Interfaces com o Usuário, Modelo Representacional de Interface com o Usuário, Orientação a Objetos

AGRADECIMENTOS

Acho que não preciso dizer o porquê nem o quanto.

Noêmia! Noêmia, Noêmia... Heuser Newton & Carmen Deus Eduardo
João "Joe" Marília Miriam & Miguel márcio
Flavinho Ivy Édson S.
Frainer "o Severo" Edward Miranda VII Fernando S. André Lemos
Melga Paulo F. Adauzinho Norberto "Tai Chi"
Nelsa Marinha... João Gluz & Lilirina J. S. Bach &
"tchê" Fábio Alfredo & Gis
Bibliotecárias (Tânia, Margarida et alli)
André Hasdrich Maurício "O Grupão" Richard
Fernando L. Matos Xda & Beth Jovelino (chefe DCEC) Jorge L. Borges
Luiz Fachini Comitê de Leitura (Weber & Tom
André Dyck Friedrich Rocha Costa
Amauri & Ângela Ítalo & família
Felipe Q.
Jorge & B. L. Moraes
Flávio F.
Fernando R.
Cláudia B. & Roberto
PARA TODOS QUE TORCERAM!
∀p | torceu(p) V

SUMÁRIO

LISTA DE ABREVIATURAS

Canonicus	Modelo CANONico para representação de Interface Com o USuário
FIU	Ferramenta para Desenvolvimento de Interface com o Usuário
IU	Interface com o Usuário
MRIU	Modelo para Representação de Interface com o Usuário
PDOO	Paradigma de Desenvolvimento Orientado a Objetos
MRTE	Modelo de Redes de Transição de Estado
ME	Modelo de Eventos
MGLC	Modelo de Gramáticas Livres do Contexto
LDIU	Linguagem de Definição de Interface com o Usuário
SGIU	Sistema de Gerência de Interface com o Usuário
MS-W	MicroSoft Windows
LOO	Linguagem Orientada a Objetos
E/S	Entrada e Saída
DA	"Desk Acessory"

LISTA DE FIGURAS

RESUMO

Interação homem-máquina, também difundida com o nome de Interface com o Usuário (ou simplesmente interface), é uma área de pesquisa relativamente recente e evidentemente multidisciplinar.

Um importante critério para projeto de interfaces é a separação de um programa interativo em seu componente computacional (aplicação) e seu componente de diálogo (que implementa a interface). Esta separação, denominada independência de diálogo, cria o papel do projetista de interfaces separado do programador da aplicação e a necessidade de novas comunicações entre os componentes do programa e o usuário.

O componente de diálogo é usualmente construído usando-se alguma Ferramenta para Desenvolvimento de Interfaces com o Usuário (abreviadas FIUs) para definição e manipulação de interfaces.

As FIUs comercialmente disponíveis atualmente (na sua maioria "toolboxes" como MicroSoft Windows e Macintosh Toolbox, entre outras), no entanto, não são tão facilmente utilizáveis, contendo literalmente centenas de rotinas e confundindo freqüentemente os papéis do projetista de interfaces e do programador da aplicação. Isto acarreta prejuízos à almejada independência de diálogo.Além disto, devido às idiossincrasias de cada FIU, o programa interativo é desenvolvido direcionado para o uso de uma FIU específica, necessitando de uma série de reformulações em caso de mudanças de FIU.

O objetivo da dissertação é a proposta de uma FIU Canônica que permite:

a) uma definição de interface de maneira mais adequada aos usuários projetistas, mais notadamente ao programador da aplicação; e

b) a portabilidade de programas interativos entre diferentes FIUs.

O componente principal da FIU Canônica é o seu modelo representacional orientado a objetos, o Canonicus, que contém as abstrações necessárias para o uso adequado dos usuários projetistas.

A portabilidade vem do fato da FIU Canônica ser, na verdade, uma camada intermediária entre a aplicação e uma FIU. Sua implementação consiste na tradução de seus objetos e operações para objetos e operações de alguma FIU subjacente.

Nesta dissertação são apresentados a arquitetura da FIU Canônica e o seu modelo representacional Canonicus assim como a sua implementação sobre duas FIUs tipo "toolbox" comerciais, o MicroSoft Windows e o Macintosh Toolbox.

PALAVRAS CHAVE: Interface com o Usuário, Ferramentas para Desenvolvimento de Interfaces com o Usuário, Modelo Representacional de Interface com o Usuário, Orientação a Objetos

ABSTRACT

Human-computer interaction, also named user interface, is a multidisciplinary and relatively recent research issue.

An important criteria to user interface design is the separation of interactive program in two components: computational component (application) and dialogue component (which implements the user interface). This separation, named dialogue independence, creates the user interface designer role independent of application programmer role and new components-user communications.

The dialogue component is usually constructed by using some User Interface Development Tool (abreviated FIU) to both user interface definition and manipulation.

The comercial FIUs available (most of them are toolboxes like MicroSoft Windows and Macintosh Toolbox), however, are often not so easily usable, since they contain literally hundreds of procedures and they confuse the interface designer and application programmer roles. Thus the desirable dialogue independence is prejudiced.

Furthermore, an interactive program is developed directed to use only one specific FIU, since each FIU has its idiosycrasies. In case of FIU change, several reformulations are needed.

The dissertation goal is the purpose of the Canonical FIU. The Canonical FIU allows:

a) an user interface definition in more adequate way to its designer-users, more notably the application programmer; and

b) interactive programs portability between diferent FIUs.

The Canonical FIU main component is its object-oriented representational model, the Canonicus, which contains the needed abstractions to user interface designers.

Portability is obtained since the Canonical FIU is an intermediate level between the application and a FIU. The Canonical FIU is implemented by a translation mechanism, mapping its objects and operations to some subjacent FIU's objects and operations.

In this dissertation, the Canonical FIU architecture, its representational model Canonicus and its implementations over two FIUs (MicroSoft Windows and Macintosh Toolbox) are presented.

KEYWORDS: User Interface, User Interface Development Tools, User Interface Representational Models, Object Orientation

1 INTRODUÇÃO

1.1 Motivação

Os fatores que mais contribuem para a difusão do uso do computador são, segundo /SMI 83/, o seu custo reduzido, o aumento de sua funcionalidade e o progresso no modo de comunicação com as pessoas. Os dois primeiros são necessários mas não suficientes. Custo reduzido, por exemplo, permite que as pessoas **comprem** computadores enquanto uma comunicação adequada permite que as pessoas **usem** os computadores.

Isto leva-nos a crer que as possibilidades do uso do computador nas mais variadas áreas não são limitadas, primordialmente, pelo seu poder de computação, mas também pelo seu poder de comunicação com as pessoas. Esta comunicação é denominada interação homem-computador ou simplesmente interface com o usuário (abreviada IU).

Em relação ao estudo de outros tópicos de desenvolvimento de software, a pesquisa de IUs é relativamente recente. A área de IU é evidentemente multidisciplinar, combinando métodos experimentais dos psicólogos educacionais e estudiosos de ergonomia e de fatores humanos com o ferramental dos cientistas de computação. Por isto, a construção de IU tem dois problemas inerentemente ligados:

a) Como construir uma boa interface (associado à definição do que é uma "**boa**" interface), que é um problema eminentemente de fatores humanos;

b) Como prover um ambiente no qual boas interfaces possam ser construídas, que é um problema eminentemente computacional.

O alto custo de um sistema interativo resulta em grande parte do desenvolvimento "ad hoc" dos componentes responsáveis pela interface. John Page, o criador do PFS:File, estima que 50% do código do produto foi devotado à IU e que se houvessem ferramentas de auxílio o tempo de desenvolvimento seria menor /CAR 83/. A motivação para o surgimento de ambientes de desenvolvimento de IU é a mesma que impulsionou a pesquisa em ambientes de desenvolvimento de software: o auxílio ao processo de desenvolvimento ("Casa de ferreiro, espeto de FERRO"), visando a diminuição do esforço e do tempo do processo de produção de software (IU) e a melhoria da qualidade do software (IU) produzido.

O desenvolvimento de IU é então altamente experimental, envolvendo um ciclo iterativo de projeto/avaliação/revisão, ou seja, com a IU sendo sucessivamente construída e submetida à avaliação, visando alcançar a satisfação do usuário.

Um efetivo suporte a estas tarefas é conseguido com a utilização de ferramentas denominadas Ferramentas para Desenvolvimento de Interface com Usuário (abreviadas **FIU**) /MYE 89/.

Um importante critério para FIUs é a separação de um programa interativo em seu componente computacional

(aplicação) e seu componente de diálogo (interface). Esta separação cria o papel do projetista de interfaces separado do programador da aplicação e a necessidade de novas comunicações entre os componentes do programa e o usuário.

O processo de desenvolvimento de uma IU usando uma FIU envolve a representação dos seus componentes e das formas de diálogo utilizadas. Isto é feito através de modelos de representação de IUs. Implícita ou explicitamente, toda FIU está vinculada a modelos de representação de IU.

As FIUs comercialmente disponíveis atualmente (MicroSoft Windows e Macintosh Toolbox, entre outras), apesar de tornarem o uso do computador mais adequado ao usuário final, não são facilmente utilizáveis pelos desenvolvedores de programas interativos. Isto acontece porque elas lidam com muitos detalhes a uma só vez e confundem freqüentemente os papéis do projetista de interfaces e do programador da aplicação.

Uma conseqüência disto é que a aplicação é desenvolvida direcionada para uma FIU específica, dificultando sua portabilidade para outros ambientes (usando possivelmente outras FIUs).

A solução para este problema é usar uma representação de mais alto nível para a IU e definir uma FIU que use esta representação. Esta dissertação investiga a definição desta FIU, determinando sua arquitetura e seu modelo de representação de IUs.

A FIU proposta, denominada **FIU Canônica**, é implementada através da tradução de todos seus objetos e operações para objetos e operações de alguma FIU comercial subjacente (no caso Macintosh Toolbox e Microsoft Windows), permitindo seu uso em ambientes para os quais esta tradução é feita.

A FIU Canônica é desenvolvida segundo o paradigma de orientação a objetos e seu modelo de representação possui mais abstrações do que os modelos das FIUs subjacentes, pretendendo prover portabilidade de programas interativos e suporte mais adequado aos usuário projetistas, mais notadamente o programador de aplicação.

1.2 Objetivos

O objetivo principal desta dissertação é a proposta de uma FIU Canônica , mapeada para FIUs comerciais (no caso, Mac Toolbox e Microsoft Windows) através de mecanismos de tradução. Esta proposta inclui a definição da arquitetura da FIU e do seu modelo de representação de IU.

Como metas secundárias tem-se:

a) a pesquisa de arquiteturas e de modelos de representação de IUs e a determinação da abordagem de modelagem a ser utilizada para a FIU Canônica; e

b) o estudo das FIUs Apple Macintosh Toolbox e o Microsoft Windows, com a explicitação de suas características;

Como pano de fundo da dissertação, mas não menos importante, está a investigação e sistematização da área de IU, um assunto de pesquisa recente no ambiente acadêmico local(UFRGS) e nacional.

1.3 Estrutura da dissertação

O capítulo 2 apresenta alguns conceitos e definições sobre IUs, ressaltando-se a noção de modelos de representação de IUs.

O capítulo 3 apresenta as Ferramentas para Desenvolvimento de Interfaces com o Usuário (FIUs), seus usuários, suas funções, suas arquiteturas e seus tipos. Duas FIUs comerciais (Mac Toolbox e MS-Windows) são detalhadamente descritas.

O capítulo 4 consiste da análise de alguns problemas das FIUs descritas, que levaram à proposição da FIU Canônica, e da explicação das características da FIU Canônica.

No capítulo 5 são investigadas as características do Paradigma de Orientação a Objetos que levaram à sua utilização neste trabalho assim como características do enfoque de Orientação a Objetos adotado.

O capítulo 6 é a proposta da arquitetura da FIU Canônica.

O capítulo 7 é a proposta do modelo de representação de IUs da FIU Canônica: o modelo Canonicus.

No capítulo 8 é apresentado um exemplo de uso da FIU Canônica.

O capítulo 9 descreve enfoques de tradução da FIU Canônica para as FIUs descritas.

No capítulo 10 são tecidas algumas observações conclusivas e sugeridos alguns tópicos de pesquisa futura.

A dissertação possui, além disto, 6 apêndices:

A - Elementos, Operações e Eventos do Macintosh Toolbox

B - Elementos , Operações e Eventos do MicroSoft Windows

C - Classes Genéricas da FIU Canônica

D - Classes Concretas da FIU Canônica para a Mac Toolbox

E - Classes Concretas da FIU Canônica para a Microsoft Windows

F - Gramática usada para a notação do Canonicus

2 INTERFACE COM O USUÁRIO

"The user interface is the program"
Alan Kay

2.1 Introdução

Neste capítulo são apresentados conceitos e definições relativos à interface com o usuário.

Um programa interativo é formado de componentes computacionais (doravante denominados **aplicação**) e de componentes de diálogo. Programas interativos muitas vezes combinam estes componentes num bloco monolítico, misturando decisões sobre diálogo e estilos de interação com detalhes procedurais do programa, o que os torna extremamente difíceis de manter e modificar.

Idealmente, os termos "interface com o usuário" e "componente de diálogo de um programa interativo" são definidos separadamente para denotar respectivamente a comunicação entre um usuário humano e um sistema computadorizado (o primeiro) e o meio para esta comunicação (o segundo). A literatura tem indistintamente cunhado o termo IU a ambos conceitos. Nesta dissertação, IU é a troca de símbolos e ações entre o homem e o computador enquanto componente de diálogo é o software que dá suporte para realização desta troca. A IU estabelece as regras para a comunicação entre o usuário e os programas, ou seja, é a representação externa das capacidades do programa. O usuário "sente" o programa através da IU.

Da pesquisa sobre desenvolvimento de IUs têm surgido vários conceitos novos, como **metáforas** e **independência de diálogo**, que já fazem parte do vocabulário da área. No entanto, esta ânsia por novos conceitos e novas experiências muitas vezes ofusca o (re)aproveitamento de conceitos já existentes na computação e que também podem ser adequados à construção de IUs, como o conceito de **modelo**. Desenvolver uma IU para um programa é modelar uma IU para o programa.

Os dois conceitos novos citados e uma explicação da noção de modelos de representação de IUs são dados a seguir.

2.2 Independência de diálogo

O desenvolvimento de interfaces de qualidade ("boas" interfaces) envolve um ciclo de projeto e avaliação de IUs. Um importante critério, então, para uma FIU é a facilidade (e rapidez) de modificação das IUs desenvolvidas.

Os pesquisadores de Banco de Dados encontraram um problema similar na necessidade de fácil alteração dos dados sem a alteração do programa correspondente e a solução encontrada foi a independência de dados. Um conceito análogo surgiu na área de IU para a construção modular de programas interativos: a divisão funcional entre componentes computacionais (aplicação) e componentes de diálogo (IU), que é convencionalmente denominada **independência de diálogo**. Na prática, esta independência

faz com que as decisões de projeto que afetam a IU sejam isoladas das que afetam a estrutura da aplicação /HAH 89/.

A separação entre a aplicação e o componente de diálogo resulta em dois benefícios essenciais:

a) a IU pode ser projetada por um especialista na área, denominado **projetista de interface**, que pode usar conhecimentos de psicologia, cognição, fatores humanos, aprendizagem, ergonomia e artes gráficas; e

b) o componente de diálogo e a aplicação podem evoluir (serem modificados) separadamente;

A separação do componente computacional do componente de diálogo cria novas comunicações entre estes componentes e o usuário: o diálogo externo e o diálogo interno, conforme figura 2.1.

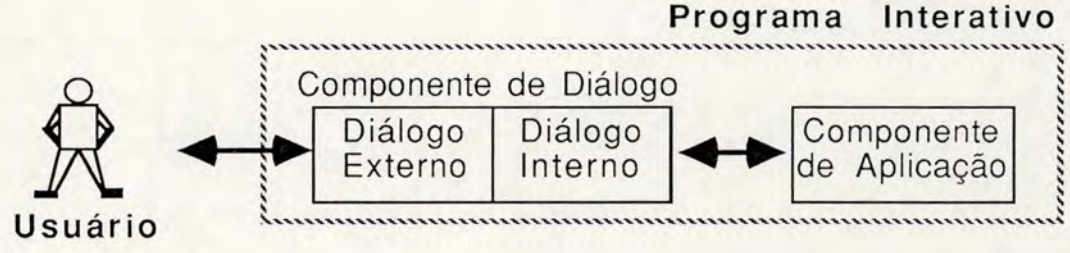

Figura 2.1 - Programa Interativo e seus componentes

A interação entre o usuário final e o componente de diálogo é denominada de **diálogo externo.**

A comunicação entre o componente computacional e o componente de diálogo é denominada **diálogo interno.** Este diálogo é a base para a comunicação entre o projetista da aplicação e o projetista da interface a tempo de projeto.

Diálogo interno não é "visto" a tempo de execução mas sua representação formal a tempo de projeto é a chave para a independência de diálogo. A IU e a aplicação podem ser mudadas sem afetar uma a outra, desde que mantenham consistência com a representação do seu diálogo interno.

Considerando esta separação, uma IU pode ser definida como o conjunto de regras capaz de gerenciar os protocolos para interação (diálogo externo) e o controle da execução das funções da aplicação (diálogo interno) /BEN 86/.

2.3 Metáforas de diálogo

De acordo com /HAH 89/, há ao menos duas metáforas que descrevem os modos pelos quais os homens interagem com o computador: a metáfora conversacional e a metáfora espacial.

A maioria das FIU tem tradicionalmente sido projetada para suportar a **metáfora conversacional** (também chamada diálogo seqüencial ou modal), onde a IU é vista como uma conversação entre o usuário e o sistema (daí originou-se o uso do termo "diálogo" como sinônimo de IU). O usuário expressa ações a serem executadas em alguma linguagem e o sistema responde. O usuário então avalia o

resultado e expressa novos comandos visando atingir algum objetivo. Esta metáfora, implícita ou explicitamente, faz uso de um modelo de linguagem para Entrada (input) e de outro para a Saída (output), respectivamente **linguagem de ação** e **linguagem de apresentação**, na nomenclatura proposta por /BEN 86/. Três aspectos destes dois modelos de linguagem precisam ser considerados: os itens léxicos, a sintaxe e a semântica, que dizem respeito respectivamente, à formação de itens de entrada/saída (E/S), à combinação de itens formando sentenças de E/S e à invocação de funções da aplicação em resposta a comandos do usuário. Esta metáfora inclui estilos de interação como Pergunta/Resposta ("question-answer"), Linguagens de Comandos, Formulários ("Form-fill") e navegação em uma rede de Menus.

O uso da **metáfora espacial** (também chamada diálogo assíncrono ou não modal) é mais recente. Com esta metáfora, o usuário tem a ilusão de estar agindo no seu espaço de trabalho sobre os objetos de interesse **sem** a intermediação do sistema. Isto é contrastado com a metáfora conversacional onde os objetos de interesse são referidos abstratamente (p.ex. por nome), seu estado é exibido apenas quando solicitado e o usuário manipula objetos apenas através de uma linguagem de ações. O estilo de interação intimamente associado a esta metáfora é a Manipulação Direta em que o usuário mostra o que fazer "pegando" e manipulando ("snap and dragging") representações visuais de objetos /SHN 87/. A interação é executada tipicamente com o uso de "mouse" (ou algum outro dispositivo de E/S que permita apontamento e movimentação na tela).

Apesar de ambas as metáforas serem utilizadas hoje em dia, a tendência é o uso crescente da metáfora espacial. Esta tendência é um dos mais significantes

fenômenos na área de IU hoje e é exemplificada pela difusão do computador pessoal Apple Macintosh /GRE 84/.

A sintaxe de uma IU com manipulação direta é dada em termos de objetos individuais e envolve mais ações físicas como "apontar" e "arrastar" do que conceitos lingüísticos. Isto implica que as FIUs baseadas em transições de estados ou gramáticas (ver próxima seção) não são bons candidatos para suportar manipulação direta. Uma maneira de incrementar a comunicação é prover representações de objetos na tela que reflitam o comportamento dos objetos do domínio da aplicação, o que é convencionalmente denominado "feedback semântico". Fazendo com que as representações na tela dos objetos reflitam as alterações dos objetos das aplicações, consegue-se a ilusão de que a representação do objeto **é** o objeto.

2.4 Modelos de Representação de IUs

A gerência de IUs, do ponto de vista da Ciência da Computação, focaliza o processo de desenvolver interfaces de qualidade ("boas" interfaces) e inclui tarefas de representação, projeto, implementação, execução, avaliação e manutenção de IUs. Como pano de fundo destas tarefas, está o modo pelo qual uma IU é compreendida, o que influencia o modo de executarmos e relacionarmos estas tarefas. O conceito de IU, dado acima, como a comunicação entre o usuário e a aplicação, leva a considerar uma IU como um exemplo típico de um **sistema**, na concepção de sistema dada por /COH 86/.

Um sistema tem como características principais:

- **composição** ou capacidade de possuir estrutura interna; e

- **ação** ou capacidade de exibir comportamento.

Modelar uma IU é então a atividade de criar modelos para descrever a sua composição e ação.

Pode-se criar diferentes modelos para descrever IUs. Alguns modelos incluem os processo perceptivos e cognitivos do usuário /MOR 81, DEH 81/, como os modelos de tarefas. Embora o entendimento destes processos seja importante para o projeto de uma "boa" interface, neste trabalho os processos mentais do usuário são separados da IU propriamente dita. Similarmente, os processo internos à máquina - as computações que são "opacas" ao usuário - não são incluídas nesta dissertação.

A abordagem de modelagem utilizada nesta dissertação usa um Modelo de Representação de Interface com o Usuário (abreviado **MRIU**) para descrever a IU.

MRIUs são esquemas notacionais para especificar instâncias particulares da interação homem-máquina, ou seja, formalismos para descrever detalhes de forma, conteúdo e seqüenciamento das partes de uma IU específica, tanto no que diz respeito ao diálogo externo quanto ao interno.

Um MRIU é necessário para registrar/documentar o processo de desenvolvimento da interface, representando informações dos aspectos visíveis (apresentação ou aspectos externos) e não-visíveis (diálogo e composição ou aspectos

internos) da IU. O processo de desenvolvimento de uma IU pode ser decomposto em duas partes:

- a parte estática da IU (correspondente à composição do sistema), na qual se decide quais elementos de IU (janelas, menus, ícones, etc.) se irá utilizar e como será a sua apresentação visual (cores, formatos, estilos, etc.); e
- a parte dinâmica (correspondente à ação do sistema), na qual se decide qual e como será a interação e como serão os diálogos interno e externo.

Idealmente, um MRIU deve:

a) ter uma base formal;

b) ser completo na sua habilidade de representar os aspectos estáticos e dinâmicos de IUs; e

c) ser independente das ferramentas implementadas a ele associadas.

Três dos principais MRIUs existentes na literatura são /GRE 86, SHN 87, MYE 89/ : o modelo de Redes de Transição de Estado (MRTE), o modelo de Eventos (ME) e o modelo de Gramáticas Livres do Contexto (MGLC).

O MRTE é baseado em diagramas de transição de estado, que consistem de um conjunto de **estados** e um conjunto de **ligações**, as quais determinam como é a transição de um estado a outro. Na sua forma mais simples para descrição de IUs, cada ligação é rotulada por um "token" de entrada que representa uma ação executada pelo usuário. O diálogo move-se de um estado para outro se

existe uma ligação entre estes dois estados rotulada por um "token" de entrada. Esta forma descreve, então, as seqüências de ações que o usuário pode executar mas não descreve as ações geradas em resposta pelo computador.

Estas ações podem ser associadas, numa forma adicional de descrição, às ligações do diagrama. Quando há uma transição entre estados, a ação é executada. Em alguns casos, pode-se também associar à ligação a saída a ser exibida como "feedback" semântico da ação. Extensões do MRTE, como redes de transição recursivas, que permitem diagramas recursivos, e redes de transição aumentadas, que além de estados e ligações possui um conjunto de registradores, são explicadas com mais detalhes em /GRE 86/.

Os principais componentes do Modelo de Eventos são os **eventos** e os **manipuladores de eventos.** Os eventos são inicialmente gerados pelos dispositivos de entrada durante a interação com o usuário. Quando um evento é gerado, é enviado a um ou mais manipuladores. Um manipulador de evento é um processo capaz de tratar certos tipos de eventos pela execução de uma rotina. Esta pode executar alguma computação, gerar novos eventos ou chamar rotinas da aplicação. Conceitualmente, todos manipuladores de eventos da IU são concorrentes, processando os eventos conforme chegam. Um manipulador pode somente processar um evento de cada vez, de modo que pode ser encarado como um monitor. Os eventos, na prática, são colocados, quando gerados, em uma fila para serem processados na ordem em que ocorreram.

O MGLC clássico é baseado em **gramáticas livres do contexto** estendidas para a invocação de ações semânticas. Suas características principais são:

- os terminais da gramática são os "tokens" de entrada que representam as ações do usuário;
- as produções da gramática definem a linguagem empregada pelo usuário em suas interações com a aplicação;
- ações semânticas (da aplicação) são associadas às produções da gramática e são executadas quando a produção é usada para o reconhecimento de uma entrada do usuário.

Um MRIU que é uma evolução do MGLC é o modelo GRAEDIUS, que utiliza gramática de atributos (estendida) para a definição da IU /SPP 90/, substituindo as ações semânticas do MGLC por equações semânticas envolvendo o cálculo de atributos e chamadas de funções de interação.

Detalhes sobre GRAEDIUS podem ser encontrados em /SCH 91/.

Qualquer MRIU, independentemente de sua base formal, pode usar uma notação gráfica ou textual para representar uma IU. Uma notação textual (linguagem) para tal fim é denominada **LDIU** (Linguagem de Definição de Interface com o Usuário). As notações (gráficas ou textuais) lidam em geral com dois aspectos: um relativo ao diálogo externo e outro relativo ao diálogo interno. Muitos pesquisadores têm desenvolvido políticas e linguagens

direcionadas para o primeiro aspecto (veja, por exemplo /MAR 90/) mas pouco tem sido feito para o segundo.

Esta dissertação visa propor um MRIU para a descrição do diálogo interno de uma IU, ou seja, um modelo de descrição para a comunicação IU-aplicação.

3 FERRAMENTAS PARA DESENVOLVIMENTO DE INTERFACES COM O USUÁRIO (FIUS)

> "Todos os homens, no vertiginoso instante do coito, são o mesmo homem. Todos os homens que repetem uma linha de Shakespeare são William Shakespeare".
>
> Jorge Luis Borges

Neste capítulo são apresentadas as Ferramentas para Desenvolvimento de Interfaces com o Usuário (FIUs) e seus usuários. A introdução contém um pequeno histórico das FIUs. A seguir funções, arquiteturas e os tipos de FIUs são definidos. Finalmente, duas FIUs comerciais (o Macintosh Toolbox e o Microsoft Windows) são descritas.

3.1 Usuários de uma FIU

Há, no mínimo, quatro papéis distintos no desenvolvimento de um programa interativo. Freqüentemente um único indivíduo pode ocupar mais de um destes papéis, dependendo do escopo do projeto e das suas capacidades.

Estes papéis são /OLS 84/:

- o usuário final ou simplesmente **usuário;**
- o **projetista da interface;**
- o **analista da aplicação** e
- o **programador da aplicação** ou simplesmente programador.

O papel do usuário final é obviamente usar o programa para alcançar seus objetivos. Sua atenção primordial é então direcionada às suas tarefas (objetivos) e não aos meios pelos quais eles são alcançados.

O projetista de interface projeta e descreve a IU. Esta tarefa inclui a especificação da forma de cada interação, de modo a prover estilos de interação naturais e apropriados e gerar diálogos compreensíveis e consistentes. Para isto, o projetista de interface deve estar familiarizado com as funções da aplicação, conhecer as características do usuário final e dominar os vários estilos e dispositivos de interação disponíveis na FIU adotada /OLS 84/.

O analista da aplicação, junto com o usuário final, é responsável por definir e especificar o problema que a aplicação visa resolver. Após isto, especifica os programas a serem desenvolvidos.

O programador de aplicação produz os programas especifica/dos pelo analista, necessários para executar as funções do sistema. Os programas usam a IU definida pelo projetista de interface.

Neste trabalho, usaremos o termo **usuário projetista** para referenciar indistintamente o projetista de interface e o programador de aplicação.

Uma FIU não é totalmente adequada se não pode ser apropriadamente utilizada por todos usuários que tenciona

suportar. Os usuários de uma FIU são os desenvolvedores do programa interativo envolvidos com o uso da FIU, ou seja, os usuários projetistas. O usuário final usa a IU desenvolvida com auxílio de uma FIU.

3.2 FIU

Um princípio de comunicação efetiva homem-máquina deve abstrair os objetos e operações do domínio do usuário e construí-los no ambiente computacional de maneira consistente.

Isto fez com que os produtores de software concebessem interfaces mais eficientes e de fácil aprendizado, permitindo ao usuário concentrar-se sobre suas tarefas (e não sobre o sistema) de maneira mais natural.

Foram introduzidos nas interfaces elementos como janelas, menus, ícones, gráficos, efeitos sonoros e outros tantos que visam a construção de boas interfaces. Novos dispositivos de E/S começaram a ser utilizados como, p.ex., "mouse", canetas, "trackballs", telas sensíveis ao toque, "plotters", etc.

Estas novas tecnologias, ao mesmo tempo que facilitaram seu uso, tornaram mais complexa a tarefa de projeto e implementação do software interativo. O usuário projetista precisa utilizar vários elementos de interface e dispositivos de E/S, o que lhe obriga a conhecer e usar o tratamento adequado a cada um.

Historicamente, para facilitar o uso destes dispositivos (principalmente para funções de saída), foram criadas primitivas gráficas que dão ao projetista uma visão abstrata (dispositivo lógico ou virtual) que esconde os detalhes particulares de cada um. Um conjunto destas primitivas forma um **pacote gráfico** (GKS /GKS 84/, Core /COR 79/), cuja ênfase é no poder gráfico, raramente provendo funcionalidade suficiente para as necessidades de uma IU. Isto levou a uma nova classe de ferramentas um pouco mais orientada a IUs: **os gerenciadores de janela** (SunWindows /MYE 88/, Star /SMI 83/). Tipicamente, um gerenciador permite que apenas uma janela por vez possa estar ativa para receber entrada via teclado ou "mouse". Cada janela age como um terminal lógico separado ("viewport") com E/S própria. O usuário pode manipular as janelas (mover, reduzir/aumentar,etc.) e a comunicação entre elas é usualmente feita com o conceito de "clipboard" ("cut and paste").

Como IU são mais difíceis de construir à medida em que são mais fáceis de usar, engenheiros de software devem, mais do que estudar aspectos ergonômicos e psicológicos para a construção de boas IUs, se esforçar para o desenvolvimento de um ambiente que propicie que boas IUs possam ser construídas. Ou seja, o que se precisa são **políticas** que possam ser aplicadas durante a especificação e o projeto de IUs e **técnicas** que aliviem o usuário projetista de lidar com detalhes de implementação de IU em baixo nível.

Nesse esforço enquadram-se as Ferramentas para Desenvolvimento de Interface com o Usuário (**FIU**), que são conjuntos de rotinas ou programas que auxiliam a construção de IUs.

FIUs são construídas sobre um gerenciador de janelas, levando muitas pessoas a confundirem os dois conceitos. Há, no entanto, uma clara separação. 'E responsabilidade do gerenciador de janelas implementar e gerenciar os terminais virtuais (janelas) enquanto é responsabilidade da FIU suportar a construção de IUs dentro de cada terminal virtual.

3.2.1 Funções de FIUs

As funções de uma FIU podem ser agrupadas em:

a) funções de desenho e

b) funções de interação.

Funções de desenho são conjuntos de operações em que a manipulação gráfica **é** o objetivo da função e o seu resultado é o que é exibido na tela, ou seja, a representação gráfica exibida na tela é o objeto de interesse do programa que usa as funções.

Funções de interação são conjuntos de operações em que a manipulação gráfica **não é** o objetivo da função, mas sim utilizada para lidar com elementos típicos de IU como janelas, menus e caixas de diálogo.

Na primeira categoria estão as funções de desenho costumeiramente usadas em programas como editores gráficos, ferramentas de CAD e processadores de imagens e o seu conjunto é denominado pacote gráfico.

Na segunda categoria estão as funções necessárias para o desenvolvimento de IUs para uma série de aplicações. Muitas das funções de interação usam funções de desenho para exibição dos seus elementos de IUs, mas de modo **transparente** para quem as invoca.

Neste trabalho, serão consideradas as funções da segunda categoria, pois não há interesse em investigar/definir pacotes gráficos.

3.2.2 Arquiteturas de FIUs

Uma arquitetura de FIU descreve teórica e genericamente a estrutura das trocas entre o usuário e o computador, isto é, como a IU relaciona-se com o usuário e a aplicação. Define, em suma, a estrutura dos diálogos interno e externo.

A maioria dos arquiteturas de FIUs existentes são adequadas para a descrição de diálogos seqüenciais (metáfora conversacional) como, por exemplo, o modelo de Seeheim, a arquitetura do sistema DMS e outros /COU 85/ /PIM 90/.

O modelo de Seeheim /GRE 86/ divide uma IU em três componentes: apresentação, controle de diálogo e interface com a aplicação, conforme figura 3.1. O componente de **apresentação** contém os detalhes específicos dos dispositivos de E/S e estilos de interação e pode ser considerado como o nível léxico da IU. O componente de **controle de diálogo** gerencia o diálogo e seu

seqüenciamento, correspondendo ao nível sintático da IU, enquanto o componente de **interface com a aplicação** contém o mapeamento entre a IU e a aplicação (interface IU-aplicação) e manipula as chamadas aos procedimentos semânticos da aplicação e seus retornos. Este componente corresponde ao nível semântico da IU. Apesar da separação dos componentes muitas vezes não ser claramente visível, o modelo de Seeheim é costumeiramente a arquitetura implícita à maioria das FIUs seqüenciais descritas na literatura /CGR 87/.

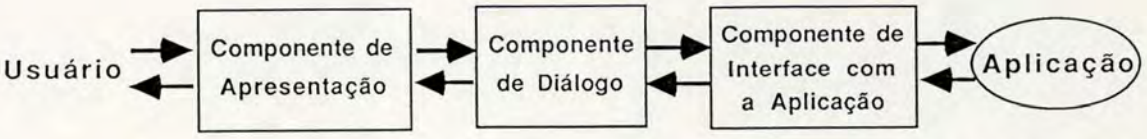

Figura 3.1 - Modelo de Seeheim

A arquitetura do sistema DMS /HAH 89/ possui também três componentes. O componente computacional contém a funcionalidade semântica da aplicação e não contém nada sobre diálogo. O componente de diálogo é composto das transações de diálogo, que incluem toda a funcionalidade, lógica e conteúdo (mensagens, apresentações, etc.) do diálogo. O componente de controle global governa a seqüência lógica entre diálogo e computação, invocando um ou outro quando necessário.

A pesquisa sobre arquitetura de FIUs para diálogos assíncronos (metáfora espacial) é ainda embrionária. Alguns modelos de pesquisa, no entanto, já existem como o Higgens /HUD 86/.

3.2.3 Tipos de FIUS

FIUs são ferramentas que suportam o desenvolvimento das funções de IU isoladas das relacionadas à aplicação. A distinção destes domínios não é direta e muitos diferentes enfoques com diferentes níveis de abstração existem, o que leva a uma classificação de FIUs.

FIUs podem ser divididas a grosso modo em duas classes principais: "toolboxes" e sistemas genéricos /KOI 88/. Alguns exemplos de FIUS estão presentes na figura 3.2.

"**Toolbox**" (do inglês "User Interface Toolbox") é uma biblioteca de funções e subrotinas de interação que podem ser chamadas pelo programa de aplicação. Conforme T. Takahashi, este é o sentido literal da "toolbox": "uma caixa de ferramentas, em que a escolha das ferramentas, o uso pretendido para cada uma delas e a seqüência do controle estão totalmente a cargo do projetista" /TAK 90/. Exemplos típicos são o Mac Toolbox e as MS-Windows Routines.

"Toolboxes" consistem de blocos de construção que são combinados de acordo com o "know-how" do projetista de interface. Os blocos de construção variam de gerência de baixo nível da estação de trabalho - como manipulação de

janelas e gráficos - ao alto nível de manipulação de diálogo. Embora "toolboxes" ofereçam muita flexibilidade e liberdade, elas deixam o projetista de interface esforçar-se sozinho para o arranjo correto dos blocos de IU necessário a criar "boas" IUs. Desde que o trabalho tem que ser refeito a cada nova aplicação, a consistência entre diferentes IUs pode ser colocada em risco.

"Toolboxes"	Sistemas Genéricos	
	"Toolkits"	SGIUs
MacToolbox /APP85/ **MS Windows**/JAM87/ **X Lib**/SCG86/ **NeWS**/PET89/	**Mac App**/SCH86/ **MVC** /GOL83b/	**ADM** /HAH89/ **Menulay** /HAH89/ **HutWindows**/KOI88/

Figura 3.2 - Exemplos de FIUs

Enquanto "toolboxes" multiplicam o esforço colocado no projeto de políticas de IU, os sistemas genéricos tentam tirar vantagem do trabalho realizado. Sistemas genéricos podem ser divididos em dois tipos: "toolkits" e SGIUs.

Um **"Toolkit"** (do inglês "User Interface Toolkit"), também conhecido por "software framework" ou "application framework", é uma estrutura de programa (esqueleto) que trata das funções de interface com o usuário e permite a adição dos componentes específicos da

aplicação nas lacunas do esqueleto. O sentido literal de um "toolkit" é, segundo T. Takahashi, que "as ferramentas incluídas têm um papel pré-definido muito claro e expresso em uma receita que acompanha o material", a qual diz "em que ordem as tarefas devem ser feitas, onde estão as peças e os encaixes e qual ferramenta utilizar em cada instante" /TAK 90/. Exemplos típicos de "toolkits" são o MVC do Smalltalk /GOL 83b/ o MacApp da Apple /SCH 86/.

Um "toolkit" consiste de um esqueleto que contém a maioria dos componentes reusáveis da IU. O projetista de interface apenas preenche as lacunas do esqueleto ou substitui as partes inadequadas para as propostas da aplicação. "Toolkits" são um passo adiante de "toolboxes".

Um **SGIU** (Sistemas de Gerência de Interface com o Usuário) é um conjunto integrado de ferramentas de suporte ao projeto, prototipação, execução, controle, avaliação e manutenção de IU /HAH 89/.

SGIUs objetivam não só oferecer uma coleção de ferramentas para o projetista de interface mas também tomar a responsabilidade do desenvolvimento e operação da IU. Então, em adição a um kernel a tempo de execução ("runtime") similar a "toolkits", SGIUs também tem ferramentas para gerar uma IU automaticamente baseada em uma especificação de mais alto nível. Infelizmente, o conceito de SGIU é ainda vago e muitos enfoques mutuamente inconsistentes existem. Como exemplos de SGIUs tem-se o SINGRAPH /OLS 83/ e o GWUIMS /SIB 86/.

Uma diferença fundamental entre estas classes de FIUs que auxilia a identificar uma FIU como pertencente a uma ou outra classe é a **localização do controle** sobre a seqüência e sincronização dos eventos durante a execução do programa interativo. Este controle pode ser determinado pela aplicação (**controle interno**) ou pela FIU (**controle externo**). Com controle interno, a FIU toma a forma de uma biblioteca de rotinas de IU, que são ativadas sob controle do programa de aplicação. Com controle externo, a FIU assume total responsabilidade pela operação da IU e chama rotinas da aplicação para executar tarefas semânticas do programa. Naturalmente pode haver um controle misto, chaveado entre a FIU e a aplicação mas seu mecanismo é muito complexo e não difundido. Usando-se uma "toolbox" o controle é interno, enquanto "toolkits" e SGIUs possibilitam o controle externo.

Em suma, o uso de FIUs tem possibilitado a criação de IUs mais adequadas, facilitando sua implementação e tornando mais econômica sua manutenção. FIUs proporcionam, também, um efetivo auxílio ao desenvolvimento de programas, pois:

a) provêm uma comunicação consistente entre o programa e o usuário; e

b) isolam a aplicação das complexidades do ambiente físico de E/S.

3.2.4 Descrição de FIU comerciais

As FIU comercializadas hoje em dia classificam-se em quase sua totalidade como "toolboxes". Algumas raras, como o MacApp, são legitimamente "toolkits". Não há SGIU

comerciais pois nenhuma FIU com este rótulo possui todas as funções atribuídas a SGIUs. Estes são ainda protótipos de pesquisa.

Embora com apresentação e modos de utilização diferentes, as FIU comerciais apresentadas aqui (Mac Toolbox e MS-Windows) apresentam muitas semelhanças do ponto de vista da programação.

Para uma melhor definição de seus elementos gráficos, os programas implementados nestas FIUs necessitam de arquivos auxiliares, denominados **recursos.** Os recursos contém, entre outros, as definições de ícones, de menus e seus itens e de caixas de diálogo. Estas definições são preparadas graficamente através de programas utilitários das FIU que convertem estes símbolos em estruturas de dados da linguagem de programação utilizada (normalmente Pascal ou C). Estes utilitários são chamados **editores de recursos.**

Uma aplicação incorpora as estruturas definidas acima e segue uma disciplina rigorosa imposta pela FIU para uso de suas rotinas, não devendo fazer acesso direto ao vídeo nem ao teclado. A comunicação entre FIU e aplicação segue basicamente os seguintes passos /WEB 86, JAM 87, APP 85/:

- definir e registrar (para o gerenciador de janelas associado a FIU) a classe de janelas associada à aplicação;
- criar e exibir uma janela da classe definida;

- uma vez que o gerenciador controla as operações sobre esta janela, à aplicação cabe ficar periodicamente interrogando o gerenciador para verificar se existem ações do usuário (eventos) que devem ser tratados pelo programa (**"loop" de controle principal**).

Um programa interativo consiste então de uma parte de inicialização de estruturas (a definição da janela) e de uma iteração principal para o tratamento de eventos disparados pelo usuário (normalmente com um grande "CASE" dentro da iteração).

Toda E/S deve ser realizada via rotinas da FIU, sob pena de alterar a organização da tela e o tratamento de eventos. Além disto, as FIUs apresentam alguns elementos com uso consagrado na composição de diálogos como janelas, menus, ícones, caixas de diálogo e controles, com pequenas variações de uma FIU para outra /WEB 86/.

O tipo de diálogo da Mac Toolbox e MS-Windows não é inteiramente assíncrono porque há um forte relacionamento de sincronicidade com o "loop" de controle principal, que é explicitamente definido pelo programador. Uma conseqüência significativa é o fato de, uma vez que o programa está com o controle, este controle é absoluto pois nenhum evento pode desviá-lo. 'E responsabilidade do programador responder a qualquer evento e voluntariamente devolver o controle ao "loop" principal.

3.2.4.1 Mac Toolbox

A FIU do Apple Macintosh (Mac) é denominada Macintosh User Interface Toolbox ou simplesmente **Mac Toolbox.** Ela é altamente integrada com seu hardware e a maioria do seu código está armazenado em uma ROM.

Suas rotinas são divididas de acordo com a função em grupos, nos quais estão os chamados **gerentes** ("managers") das características suportadas. Existem várias interconexões entre estes grupos. Algumas rotinas de nível mais alto chamam rotinas de níveis mais abaixo. Uma breve descrição de cada grupo é dada a seguir. Mais detalhes podem ser encontrados em /APP 85/.

O **Gerente de Recursos** ("Resource Manager") é a parte da Mac Toolbox através da qual o programa acessa vários recursos como menus, estilos de letras e ícones. Os recursos são armazenados em arquivos de recursos.

Usualmente as rotinas do Gerente de Recursos são chamadas por outras partes, de modo que a aplicação utiliza-o indiretamente. O Gerente de Recursos controla o acesso aos recursos e provê rotinas que permitem o acesso à aplicação e outras partes da Toolbox. Recursos são agrupados logicamente em tipos de recursos. Referência a um recurso é então feita através de seu tipo e de seu identificador.

O **Quickdraw** é um pacote gráfico que realiza todas as operações gráficas. O Quickdraw tem algumas habilidades

não encontradas em muitos outros pacotes gráficos, como por exemplo:

- a capacidade de definição de várias portas gráficas na tela, cada uma com coordenadas próprias;
- mecanismos de recorte ("clipping") completos;
- possibilidade de desenho "off-screen", que permite preparar uma imagem antes de exibi-la.

O Quickdraw permite desenhar pontos, linhas , texto estilizado, triângulos, retângulos, polígonos em geral, círculos, elipses, regiões e gráficos definidos pelo usuário.

O **Gerente de Fontes** ("Font Manager") dá o suporte necessário a fazer uma variedade de estilos e tamanhos de caracteres usados em textos.

Todos os eventos do programa são recebidos e interpretados pelo **Gerente de Eventos** ("Event Manager"), que estabelece comunicação entre a aplicação e o mundo fora dela. O Gerente de Eventos é que permite que as aplicações monitorem as ações do usuário. Eventos no "mouse" e no teclado são detectados pelo Gerente de Eventos, que também reporta ocorrências dentro da aplicação que podem requerer uma resposta.

Sempre que o usuário pressiona um botão do mouse, digita no teclado ou insere um disco, a aplicação é notificada por meio de um evento. Uma aplicação típica do

Mac é dirigida por eventos: ela decide o que fazer de momento em momento perguntando o Gerente de Eventos por eventos e respondendo a eles um por um da maneira apropriada.

Embora a proposta principal do Gerente de Eventos seja monitorar as ações do usuário e passá-las para a aplicação de uma maneira ordenada, ele também serve como um mecanismo conveniente para enviar sinais de uma parte da aplicação para outra. Por exemplo, o redesenho de uma janela que estava sobreposta e tornou-se exposta. O usuário também pode definir seus próprios tipos de eventos e usá-los da maneira que lhe aprouver.

Há vários tipos de eventos, dependendo de sua origem e significado. Os mais importantes são aqueles que sinalizam alguma ação do usuário (teclado, "mouse", disco) e que estão melhor descritos no Anexo A. Além destes, há os eventos para ativação e atualização de janelas, os eventos de dispositivos ("drivers" e rede) e os definidos pelo usuário.

Os eventos que esperam ser processados são colocados numa fila de eventos, onde são armazenados pelo sistema operacional. O Gerente de Eventos recupera eventos da fila para a aplicação e reporta outros eventos que não são mantidos na fila, como aqueles relacionados com janelas. Os eventos não são reportados na ordem em que ocorreram: alguns podem ter uma prioridade maior do que outros. Pode-se restringir o tratamento de determinados eventos, desabilitando-os ou interrogando o Gerente de Eventos apenas por tipos de interesse.

Os programas do Mac que usam a Mac Toolbox são dirigidos por eventos. Como mencionado antes, os programas têm um "loop" principal que repetidamente verifica se existem eventos sinalizados e então usa um comando CASE para realizar a ação apropriada para cada tipo de evento. O programa precisa responder apenas os eventos que são diretamente relacionados com suas operações. /APP 85/ dá uma série de sugestões para o tratamento adequado a cada tipo de evento.

Toda informação apresentada por uma aplicação padrão Mac aparece em janelas. Para criar janelas de diferentes tipos, ativá-las, movê-las, reconfigurar seu tamanho ou fechá-las deve-se usar o **Gerente de Janelas** ("Windows Manager"). Pode-se manipular janelas sem saber como elas se sobrepõem pois o Gerente de Janelas chama o Gerente de Eventos para avisar sua aplicação que uma janela precisa ser redesenhada. Da mesma forma, quando o usuário pressiona um botão do "mouse", a aplicação chama o Gerente de Janelas para saber em qual parte de qual janela ou sobre qual item de menu foi pressionado. O Gerente de Janelas assegura que as áreas necessárias são redesenhadas quando algum aspecto da janela é mudado.

Toda janela possui uma classe (conforme o seu conteúdo) e um tipo (conforme sua apresentação).

As classes pré-definidas são:

-Documento, janela comum da aplicação;

-Alerta, janela para caixas de diálogo e alerta;

-Aplicação, janelas criadas direta ou indiretamente pela aplicação e

-Sistema, janela onde "desk acessories" são exibidos.

Os tipos pré-definidos são DocumentProc, noGrowDocProc, rDocProc, dBoxProc, plainBox e altDBoxProc,cada um com diferentes tipos de formas, linhas de contorno e bordas /APP 85/.

Toda janela tem 2 regiões: uma região de estrutura e uma de conteúdo. A de estrutura (o contorno da janela) é desenhada pelo Gerente de Janelas, conforme o tipo da janela. A de conteúdo é uma porta gráfica onde a aplicação pode desenhar usando qualquer rotina do Quickdraw.

O **Gerente de Controle** ("Control Manager") é a parte da Mac Toolbox que lida com controles, como botões, "check boxes" e "scroll bars". Através do Gerente de Controle, as aplicações podem criar, manipular e remover controles. Um controle é um objeto na tela do Mac com o qual o usuário pode, usando o mouse, disparar ações instantâneas com resultados visíveis ou mudar o contexto (parâmetros de execução) para uma ação futura.

Os tipos pré-definidos de controles são:

- botões ("buttons"), que disparam ações;

- "radio buttons", que mudam o contexto e são mutuamente exclusivos;

- "check boxes", que mudam o contexto e não são mutuamente exclusivos;

- "dials", que exibem valores de modo gráfico e pseudo-analógico os quais, conforme o tipo, podem ser modificados; um tipo de "dial" pré-definido são os "scroll bars".

Todo controle pertence a uma janela particular e são exibidos na região de conteúdo da janela. Suas coordenadas são dadas em função das coordenadas da janela. Normalmente, estas janelas são da classe de alerta.

O **Gerente de Menus** ("Menu Manager") é a parte da Mac Toolbox que permite a criação de conjunto de menus e que gerencia a escolha de opções (itens de menus) oferecidas pelos menus.

Os títulos dos menus que podem ser usados durante a execução da aplicação estão sempre presentes numa barra de menus no topo da tela. Um menu consiste de um número de itens listados verticalmente ("cortina"). Um item pode ser um comando ou uma linha divisora de grupos de comandos. O número máximo de itens é 20. Menus sempre aparecem na frente de qualquer outro elemento na tela.

Quando um título é escolhido, o menu associado é "descortinado" ("pull down"). Conforme percorre-se as opções com o cursor, os itens vão sendo realçados ("highlighted"). Quando o usuário escolhe uma opção, o Gerente de Menus comunica à aplicação qual item foi escolhido e a aplicação executa a ação correspondente.

O **Editor de Texto** ("TextEdit") é a parte da Toolbox que manipula capacidades básicas de edição e formatação de textos em uma aplicação. Suas funções básicas são: inserção/remoção de textos, substituição/modificação de textos, seleção de texto, recorte e colagem de texto ("Cut and Paste"), paginação de texto ("scrolling") dentro de uma janela e alinhamento ("justification") de texto.

O **Gerente de Diálogos** ("Dialog Manager") é a parte da Mac Toolbox que permite a implementação das caixas de diálogo e de alerta, dois modos de comunicação entre a aplicação e o usuário. Estas caixas podem conter texto informativo ou instrucional, controles, gráficos ou campos editáveis para entrada de informação.

Quando uma aplicação necessita mais informação do usuário (para um comando ou tratamento de exceções), usa o Gerente de Diálogos para criar e apresentar caixas de diálogo e alerta e obter as respostas do usuário.

Numa caixa de diálogo, o usuário fornece a informação necessária para a execução de um comando através de textos ou de botões. A maioria das caixas de diálogo requerem que o usuário responda antes de fazer qualquer outra coisa. Este tipo de diálogo é chamado de diálogo modal porque coloca o usuário num estado ou modo tal que só pode trabalhar dentro da caixa de diálogo. Um modo é uma parte de um programa na qual o usuário entra e tem de responder para sair, restringindo as operações que podem ser executadas às disponíveis dentro da caixa. Diálogo modal torna as ações futuras contingentes a partir das anteriores e essa seqüencialidade aproxima-o do diálogo seqüencial.

Há outras caixas que não requerem que o usuário responda antes de fazer qualquer coisa: são as caixas de diálogo não modal ("modeless dialog"). De fato, uma caixa de diálogo não-modal parece uma janela da classe documento: pode ser movida, desativada, ativada ou fechada como qualquer janela documento. Como não há o conceito de modo, não há a saída da caixa: para fechá-la, deve-se fechar a janela em que ela está exibida. Caixas de diálogo não-modal não necessitam de resposta. Por exemplo, uma caixa de aviso "Programa em execução" pode aparecer para usuário enquanto o programa não termina sua tarefa, após o que a caixa é removida . A falta de seqüência e sincronismo entre as tarefas aproxima o diálogo não-modal do diálogo assíncrono.

A caixa de alerta provê as aplicações com um meio de reportar erros e dar alertas. 'E similar a uma caixa de diálogo modal, mas só aparece quando algo deve chamar a atenção do usuário ou algo está errado. Há três tipos padrões de alerta: "Stop", "Note" e "Caution", cada um indicado por um ícone específico (respectivamente um sinal de exclamação, um asterisco e um sinal de interrogação) no canto superior esquerdo da caixa. O mecanismo de alerta também usa sons diferentes para estes tipos.

Uma caixa de diálogo aparece sempre numa janela de diálogo, podendo então ser livremente manipulada como uma janela (movendo, aumentando tamanho, etc.). Similarmente, uma caixa de alertas aparece numa janela de alerta, que não possui a flexibilidade da janela anterior: tem sempre a mesma aparência e comportamento, não permitindo qualquer manipulação.

O **Gerente de Mesa** ("Desk Manager") permite a manipulação de "desk acessories" (abreviados DA) pelas aplicações. DAs são mini-aplicações que podem rodar ao mesmo tempo de uma aplicação no Mac. Existem vários DAs pré-definidos como uma Calculadora, um relógio de alarme, um calendário e outros. O usuário abre um DA através do AppleMenu (que é, por convenção, o primeiro menu da barra de menus das janelas). Quando um DA é escolhido, ele é usualmente exibido em uma janela "desktop" (uma janela do sistema) e esta janela torna-se ativa. Um DA pode ter seus próprios menus e, além disso, usar algumas funções de edição (do menu Edit, por convenção o terceiro na barra de menus) como por exemplo "Cut" e "Paste" para comunicar-se com a aplicação ou outros DAs.

O **Gerente de Recorte** ("Scrap Manager") serve para permitir trocas (via comandos "Cut", "Copy" e "Paste") de texto ou gráficos entre:

- duas aplicações ou
- uma aplicação e um DA ou
- dois DAs ou
- dois trechos de uma só aplicação.

A área intermediária de transferência é chamada "desk scrap" ou "clipboard" e em geral permite apenas pequenas quantidades de dados. A ordem e o tamanho dos dados é importante.

Além de textos e gráficos, o "desk scrap" pode também conter tipos de dados específicos da aplicação.

Os principais elementos da Mac Toolbox e suas rotinas principais para manipulá-los estão descritos no Anexo A. Um exemplo de programa usando a Mac Toolbox está apresentado no capítulo 8.

3.2.4.2 MS-Windows

A revolução no mercado de FIUs começou com o Mac Toolbox. Depois dele, diversas FIU seguiram sua iniciativa e princípios. Uma FIU que tentou trazer idéias do Mac para o mundo do PC foi o Microsoft Windows, também conhecido por MS-Windows ou simplesmente **MS-W** /JAM 87/. Neste trabalho nos basearemos nas versões 2.* (versão 2.0 e 2.1, Windows 286 e Windows 386). A descrição do MS-W a seguir compara algumas de características com as apresentadas para o Mac Toolbox. Mais detalhes sobre o MS-Windows podem ser encontrados em /JAM 87/.

Embora baseado no Mac Toolbox, o MS-W é mais limitado, não possuindo todo o ambiente gráfico do Mac, principalmente a nível de sistema operacional, e mais lento, só se tornando aceitável sobre um PC-AT /WEB 86/. Sua compensação é a criação de um ambiente multi-tarefa no PC, com a execução de programas concorrentes em janelas distintas. No entanto, as rotinas de sua "toolbox" (denominadas "MS-W Routines") fazem um uso quase equivalente ao da Mac Toolbox para menus, janelas, caixas de diálogo e interação via "mouse".

Todas as aplicações sobre o MS-W (denominadas aplicações Windows) têm a mesma estrutura básica. Esta estrutura apresenta os seguintes passos:

i) verificar se a classe de janelas especificada já está definida; se não está, definir e registrar a classe de janelas;

ii) criar e exibir uma janela da classe definida;

iii) gerenciar as mensagens do e para o MS-W.

Aplicações Windows são dirigidas por mensagens. O MS-W obtém e distribui mensagens de e para todas as aplicações que estão sendo executadas simultaneamente. À cada aplicação corresponde uma janela. Registrar uma classe de janelas é uma operação fundamental no MS-Windows para tratar a concorrência entre as aplicações do ambiente. No Mac Toolbox, onde não há preocupação com concorrência, esta amarração entre aplicações e janelas não é muito importante. Dizer que duas aplicações Windows estão sendo executadas significa dizer que duas janelas estão sendo exibidas. O MS-W traduz as mensagens, convertendo a tecla ou posição do mouse e enviando a mensagem devida à janela onde ocorreu o evento. As mensagens traduzidas são colocadas numa fila de mensagens da aplicação.

Mensagens podem ser enviadas do usuário (dispositivos de E/S), de outras aplicações ou da própria aplicação para si mesma. Pode-se pensar no MS-W como um único gerente roteando mensagens para seus destinos e enfileirando-as até que sejam processadas. O MS-W não possui portanto a organização em subgerentes do Mac Toolbox. As mensagens são todas trocadas através do gerente.

A diferença principal do MS-W em relação ao Mac Toolbox é que no MS-W a ativação de suas diferentes funções é feita via mensagens enquanto no Mac Toolbox é via

invocação de rotinas. Além disto, o controle sobre o fluxo de mensagens é centralizado no MS-W e a ativação de rotinas no Mac Toolbox é feita de modo hierárquico, com rotinas de um nível mais alto ativando rotinas de nível mais baixo, conforme a estrutura hierárquica dos subgerentes.

Cada uma das mensagens se enquadra numa categoria geral. As categorias principais são:

a) Gerência de janela, que notifica mudanças no estado da janela;

b) Inicialização, usada quando um menu ou caixa de diálogo é criado pela primeira vez;

c) Entrada, gerada a cada recebimento de entrada do usuário via teclado, "mouse", "scroll bar" ou timer;

d) Sistema, gerada cada vez que o usuário acessa o menu do sistema, "scroll bars" ou caixas de tamanho das janelas;

e) "Clipboard", usada quando as aplicações tentam usar o "clipboard" do MS-W;

f) Informação do Sistema, gerada cada vez que os recursos "system-wide" são mudados devido a uma ação do usuário ou uma ação da aplicação;

g) Janelas de Controle, quando o controle acompanha a tarefa requerida e retorna à rotina chamadora;

h) Controle de Edição, para requisitar a execução de uma tarefa específica de edição (seleção de texto, "scrolling",etc.);

i) Quadro de listagem, requisitando funções específicas do quadro de listagem;

j) Notificação, usada para indicar qual ação ocorreu;

k) Área Não-cliente, quando a ação do usuário afeta a área que não á área do cliente (regiões como molduras de janelas e barra de captura da janela).

A primeira rotina requerida para as aplicações Windows é a função **WinMain**, que serve como seu ponto de entrada para as aplicações Windows. Dentro desta função, o código invocará a rotina que define e registra a classe de janelas que é usada pela aplicação. Geralmente, a definição de uma janela especifica:

- nome da classe de janelas;
- o cursor, a barra de menus e o ícone associados à janela;
- cor de fundo;
- estilo da janela;

Uma vez que a janela está registrada, a aplicação criará sua cópia da janela via função CreateWindow.

Finalmente, a função WinMAin processa todas as mensagens do e para o MS-W. O resto das rotinas em uma aplicação Windows são definidas pelo usuário.

O MS-W também possui os arquivos de recursos, onde pode-se definir, através de editores de recursos

específicos, ícones, cursores, estilos de caracteres, menus, caixas de diálogo e controles.

Os elementos do MS-W têm muitos conceitos correspondentes aos descritos para o Mac Toolbox. Por isto, seus conceitos não serão repetidos aqui. As diferenças estão na maneira de defini-los (recursos), nas estruturas de dados para representá-los e nas operações para manipulá-los e exibi-los. Por exemplo, enquanto no Mac Toolbox controles e caixas de diálogo são definidos e tratados separadamente (por subgerentes diferentes), no MS-W a definição de uma caixa de diálogos inclui a definição de todos os seus controles associados e as operações para manipulação das caixas de diálogo incluem operações para manipulação de seus controles.

Algumas operações são muitos similares como operações sobre Menus (alterar itens, inserir itens, remover itens, etc.) enquanto outras são totalmente diferentes, como o tratamento de eventos. No Mac Toolbox, os eventos sobre controles e janelas, por exemplo, são identificados e tratados por rotinas diferentes. No MS-W todos os eventos são identificados pelo gerente e depois tratados por rotinas apropriadas.

Os principais elementos do MS-W e suas operações principais estão descritos no Anexo B.

4 RUMO A FIU CANÔNICA

"Miserrimi quippe est ingenii semper
uti inventis et nunquam inveniendis"
Hyeronimus Bosch

Neste capítulo são discutidos os principais problemas das FIUs descritas e apresentadas as características da solução proposta: a FIU Canônica.

4.1 Problemas com as FIU apresentadas

FIUs apresentadas (na verdade, a maioria das FIUs existentes) têm basicamente a mesma funcionalidade, provendo-a, no entanto, de maneiras diferentes. Devido às idiossincrasias de cada FIU, a aplicação precisa se adaptar a cada FIU particular.

A maioria das FIUs comerciais disponíveis são do tipo "**toolbox**", como MicroSoft Windows e Macintosh Toolbox. Uma vez que elas contêm literalmente centenas de rotinas, o esforço necessário para compreendê-las e usá-las é considerável. Além disto, estas rotinas devem ser adequadamente usadas pelo programador da aplicação para manipular todos os eventos de entrada (expressos em geral em baixo nível como "botão direito do mouse pressionado" e "cursor na posição 20:30") e para desenhar manipular os elementos da IU (menus, janelas, botões, etc.) As rotinas da FIU incluem tanto rotinas de alto nível (p.ex. "mover janela") quanto rotinas de baixo nível (p.ex. traçar reta), formando uma mistura duramente criticada por /LIN 89/.

Usando estas FIUs, constata-se que elas não são facilmente utilizáveis. Desenvolver programas interativos com auxílio destas FIUs é uma tarefa árdua e complexa. Entre as principais causas desta dificuldade de uso enumera-se que as FIUs:

- confundem os papéis de projetista de interface e programador de aplicação;
- exigem que os usuários projetistas lidem com muitos detalhes para realização de suas tarefas; e
- não incentivam nem propiciam portabilidade.

Por que as FIUs confundem os papéis dos usuários projetistas?

Em geral, as FIUs confundem freqüentemente os papéis do programador da aplicação e do projetista de interface, deixando ao critério do programador de aplicação muitas decisões: algumas que dizem respeito ao controle da IU (o seqüenciamento), outras que dizem respeito à apresentação da IU (cores, fontes, linhas, etc.) e outras ainda a detalhes de implementação de certas características da IU ("highlight", itens obscurecidos quando não ativos, etc.). Mesmo com o uso de arquivos de recursos, as FIUs apresentadas ainda deixam muita informação sobre apresentação e forma da IU dentro do programa. O programador da aplicação precisa lidar, além de detalhes da aplicação, com decisões/tarefas de IU. Desta forma, o programador não é isolado dos detalhes de IU. Isto acontece porque elas são "toolboxes" de baixo nível, não possuindo uma política para definição e combinação de elementos de IU, deixando a cargo do programador a execução de tarefas que deveriam ser feitas pelo projetista de interface. O

estabelecimento desta política e a tomada de muitas dessas decisões deveriam ser feitas pelo projetista de interface e não pelo programador de aplicação.

Por que lidar com tantos detalhes nas FIUs?

O excesso de detalhes para uso da FIU é decorrente do excesso de detalhes necessários para descrições das IUs. Estas descrições englobam desde o layout de tela (apresentação) e o diálogo entre o usuário e o computador até a interface entre a IU e a aplicação. Nas FIUs, a descrição destes detalhes é parcialmente feita através de recursos. Muitos detalhes adicionais são acrescentados via rotinas. Lidar com todos estes detalhes de uma só vez é uma tarefa complexa para o projetista de interfaces. Isto significa que os modelos de representação (MRIUs) usados para as descrições não possuem abstrações ou possuem abstrações de baixo nível em relação ao desejado pelos usuários projetistas.

Por que as FIU não propiciam nem incentivam portabilidade?

Portabilidade é a capacidade de uma aplicação ser transportada de um ambiente para outro, de ser suportada por múltiplos sistemas. Aplicações variadas podem residir em diferentes computadores e serem desenvolvidas ou expandidas em novos ambientes.

Um programa é, em geral, desenvolvido com sua interface direcionada para o uso de uma FIU específica. Uma mudança de FIU implica em uma série de reformulações dos

conceitos de interação e reescrita do código relativo à interface. Portanto, portar um programa de uma FIU a outra necessita da intervenção direta do seu implementador, que acaba dispendendo, às vezes, mais tempo com este tipo de detalhe que com a aplicação.

A figura 4.1 mostra uma aplicação desenvolvida para diferentes ambientes. Devido às idiossincrasias das FIUs, a aplicação precisa se adaptar a cada uma. O programador de aplicação cria diferentes versões da mesma aplicação: uma para cada FIU.

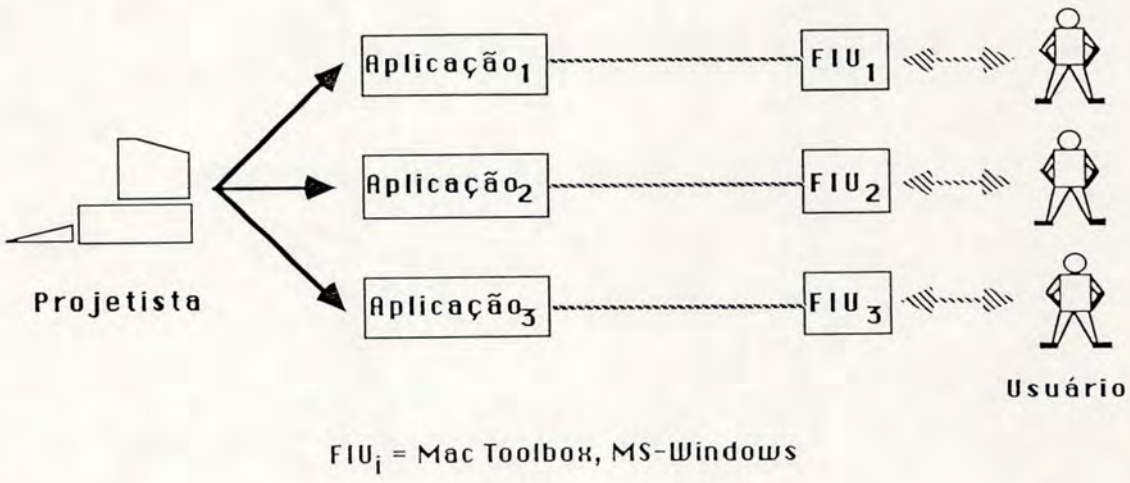

Figura 4.1 - Uma aplicação escrita para diferentes ambientes

Para a obtenção da portabilidade, basta que o diálogo interno não se altere, ou seja, que toda FIU use o mesmo protocolo de diálogo interno com a aplicação. Isto leva à necessidade de uniformização de tarefas como:

a) seleção de componentes de IU a serem usados;

b) definição dos elementos de IU e operações associadas;

c) uso dos elementos e operações definidas;

4.2 A solução proposta: uma FIU Canônica

Uma proposta para tornar as FIUs mais fáceis de usar é a criação de uma FIU canônica orientada a objetos, doravante denominada **FIU Canônica.**

A FIU proposta foi primeiramente denominada de "**canônica**" em /FPP 90/. Este nome surgiu porque esta FIU é uma coleção dos objetos e rotinas de interação mais importantes e/ou mais comuns das FIUs consideradas. Depois, constatou-se que a FIU Canônica deve ser poderosa o suficiente para permitir aos usuários projetistas a definição de quaisquer características desejadas. Desta forma, apesar do nome tradicional ser mantido, a FIU Canônica não deve ser limitada ao "mínimo denominador comum" das FIU apresentadas mas deve prever e satisfazer os anseios de seus usuários.

A FIU Canônica é **orientada a objetos.** Isto significa que toda comunicação entre IU e aplicação é estabelecida via mensagens direcionadas dos/aos objetos pertencentes a IU. A FIU Canônica, assim, possui o diálogo interno orientado a objetos - independente do paradigma ou metáfora empregados para o diálogo externo.

A Mac Toolbox, por exemplo, pode ser considerada como orientada a objetos na visão do usuário (diálogo externo) mas o seu diálogo interno é feito segundo o paradigma convencional.

A FIU Canônica é do tipo "**toolbox**", ou seja, uma biblioteca de objetos e rotinas que podem ser chamadas pelo programa de aplicação. O controle da interação é mantido a cargo da aplicação (tipo de controle interno). Este enfoque foi adotado por dois motivos:

i) a maioria das FIUs subjacentes é também "toolbox", o que torna mais imediata a correspondência entre seus componentes e os da FIU Canônica;

ii) uma "toolbox" é o passo inicial necessário à construção de FIUs mais complexas: todo "toolkit" ou SGIU é construído sobre uma "toolbox" /MYE 89/.

A FIU proposta é mapeada para as FIUs apresentadas através de mecanismos de **tradução**. Todos seus objetos e operações são traduzidos para os objetos e operações de alguma FIU subjacente. A FIU canônica permite uma visão evolutiva da IU. Pode-se usar capacidades virtualmente existentes (ou seja construídas) sobre a FIU que usamos.

A maneira pela qual uma aplicação, a FIU Canônica e a FIU específica interagem está ilustrada na figura 4.2 e apresenta as seguintes características:

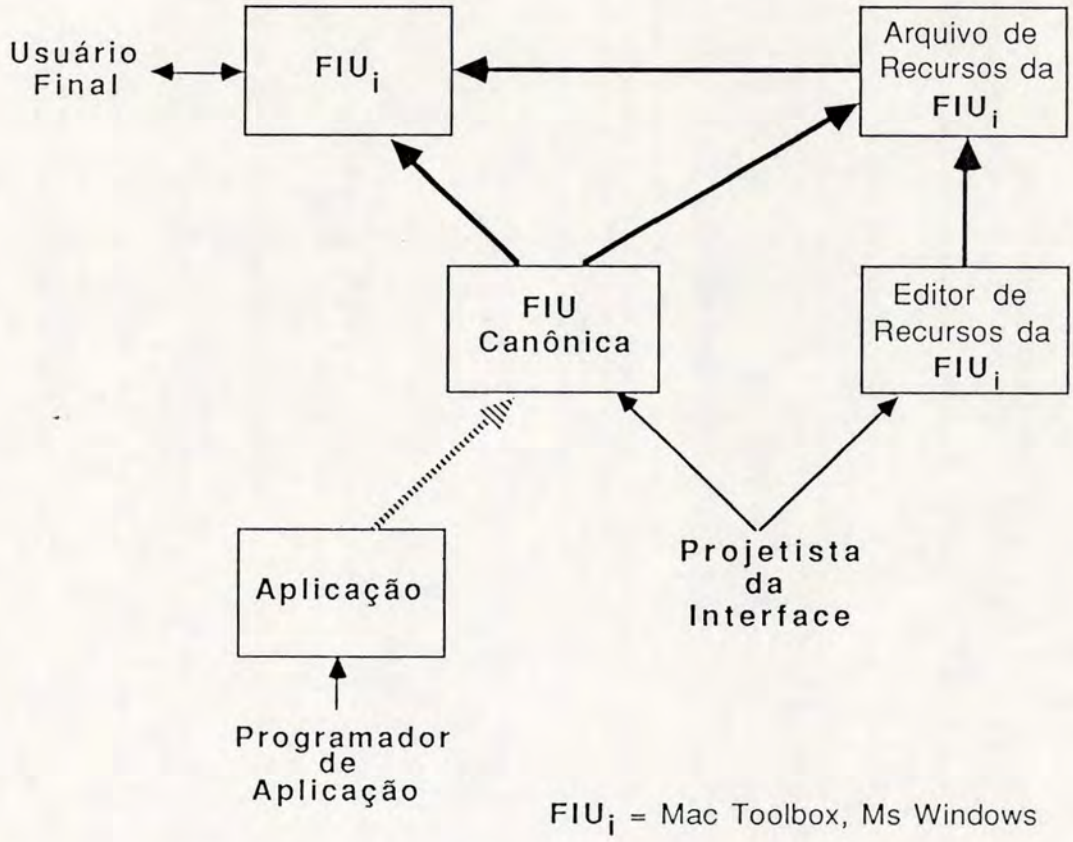

Figura 4.2 - Interação entre a FIU Canônica, a FIU específica e a aplicação

- O projetista de interface seleciona os elementos que comporão a IU. Os elementos são descritos por suas propriedades, seus relacionamentos e operações para manipulá-los (ver capítulo 6). Os elementos podem ser compostos a partir de elementos primitivos pré-definidos.

- O projetista de interface, a seguir, define as apresentações dos objetos (ver capítulo 6), podendo utilizar os editores de recursos disponíveis na FIU específica.

- A definição da IU é feita em níveis, sendo o nível abstrato o mais adequado para uso pelo programador de aplicação.

- A aplicação comunica-se com a FIU Canônica através do protocolo definido neste nível abstrato.

- O diálogo entre a aplicação e a FIU Canônica é via protocolo da FIU Canônica, com a aplicação usando a FIU Canônica para qualquer ação na IU. O diálogo real dá-se entre a aplicação e a FIU específica, uma vez que toda ação na FIU Canônica é traduzida para ações na FIU específica (ver capítulo 9). O programador da aplicação, no entanto, não precisa conhecer esta tradução, utilizando apenas o protocolo da FIU Canônica. A tradução da FIU Canônica é transparente para o programador de aplicação.

A FIU Canônica pode ser entendida como uma **abstração da FIU específica** da mesma forma que uma linguagem de alto nível é uma abstração de uma linguagem de baixo nível, escondendo do programador a complexidade da arquitetura da máquina. Esta analogia entre FIUs e linguagens de programação está expressa abaixo.

Linguagens de Programação	**FIUs**
Alto Nível (Pascal, Cobol,etc)	FIU Canônica
Baixo Nível (Assembly)	FIU específica (Mac,MSWindows)
Hardware	Dispositivos de E/S

A analogia da FIU Canônica com linguagens de programação de alto nível é total: linguagens de programação de alto nível (FIU Canônica) escondem os detalhes de baixo nível da linguagem de montagem (da FIU

subjacente) do programador ao mesmo tempo que lhe oferece estruturas de dados e comandos (elementos e operações) mais adequados para sua tarefa.

A idéia da FIU Canônica é que:

- o usuário final usa o "look & feel" (a FIU) que está habituado a usar;
- os usuários projetistas usam FIU Canônica para projetar e implementar mais facilmente as IUs para as aplicações. Esta IU projetada segundo a FIU Canônica é independente de FIU de modo que o usuário final possa usar a aplicação com a FIU que desejar;

A arquitetura da FIU Canônica, isto é, a descrição de como seus elementos são organizados e como é a comunicação da IU com a aplicação e com o usuário é apresentada no capítulo 6.

O modelo que a FIU Canônica usa para descrição de IUs é o **MRIU Canonicus.** O Canonicus lida apenas com a representação do diálogo interno da arquitetura da FIU Canônica, uma vez que o estilo de diálogo externo utilizado é o estilo da FIU subjacente utilizada e não pode ser modificado via FIU Canônica. O modelo Canonicus é apresentado no capítulo 7.

A FIU Canônica tem vários objetivos significativos, entre os quais:

a)explorar eficientemente as tecnologias de hardware disponíveis e permitir adaptação fácil a novas tecnologias;

b)suportar prototipação de IUs;

c)separar IU da aplicação, isto é, propiciar independência de diálogo;

d)servir como plataforma para gerência da definição e do controle de execução de IUs;

e)permitir a clara separação dos papéis do projetista de interface e do programador de aplicação, provendo níveis diferentes de uso e definição para cada um;

f)definir diálogos internos de IU padronizados, mais adequados à tarefa do programador de aplicação;

g)ser flexível, exigindo mudanças mínimas no caso de uma transformação em "toolkit" ou SGIU;

h)resolver as deficiências das FIUs em que se baseia.

Este último item deve ser melhor investigado.

A FIU Canônica separa os papéis do projetista de interface e do programador da aplicação

A FIU Canônica permite a separação dos aspectos da IU relativos à apresentação dos relativos ao controle de diálogo. Esta separação é adequada pois:

a)complementa a noção de independência de diálogo;

b)permite maior clareza dos papéis do projetista de IU e do programador de aplicação;

c)leva a definições separadas destes aspectos:

- apresentação: definida/usada pelo projetista de interface;

- controle de diálogo: numa "toolbox", definido e usado pelo programador da aplicação.

Na FIU Canônica, as informações sobre IU para o programador de aplicação são diferentes das informações sobre IU para o projetista de interface.

O projetista de interface define os objetos que comporão a IU e suas apresentações. Os objetos são descritos por suas propriedades , seus relacionamentos e seus operadores disponíveis. Uma apresentação define cores, formatos, estilos e tamanhos de caracteres, etc.

O programador da aplicação define quando usá-los e como associá-los, junto com seus operadores, com as funções da aplicação.

Como a FIU Canônica trata os excessos de detalhes?

A maneira efetiva de melhorar a tarefa de projeto de IU e de desenvolvimento de aplicações é prover um novo tipo de FIU, uma FIU que possua operações mais adequadas ao programador e propicie ao projetista de interface condições mais adequadas ao projeto de IU. Isto pode ser conseguido através de **abstrações** /GRE 87/.

Uma abstração de um sistema é um modelo com omissão deliberada de detalhes irrelevantes /SMI 77/. A escolha da relevância é feita considerando a intenção da abstração e seus usuários. O objetivo de usar abstrações na FIU Canônica é permitir que os usuários projetistas considerem apenas os detalhes da IU que são relevantes a suas tarefas e ignorem os outros.

Em algumas aplicações (e IU é um caso típico), um sistema pode ter muitos detalhes relevantes e fica complexo gerenciar toda esta informação. Nestes casos, a facilidade de gerência pode ser provida decompondo-se o modelo em uma **hierarquia de abstrações**, com os detalhes introduzidos de maneira controlada a cada nível da hierarquia. As abstrações de um determinado nível permitem que alguns detalhes sejam temporariamente ignorados pelo nível imediatamente superior.

Uma vantagem de tal hierarquia de abstrações é a capacidade de prover diferentes níveis de abstração (diferentes níveis de relevância de informação) a diferentes usuários. Desta forma, um único modelo pode ser compartilhado entre vários usuários sem comprometer seus acessos às suas informações relevantes. Outra vantagem da hierarquia é a estabilidade do modelo. Alterações em um nível, em sua maioria, são transparentes aos níveis superiores.

Uma FIU com hierarquia de abstrações permite diferentes visões destes níveis de abstração, com cada usuário (programador de aplicação, projetista de interface) enxergando/manipulando o nível adequado para suas tarefas. Isto teria como conseqüências:

- Para o projetista de interface : níveis de abstração diferentes permite tratar problemas diferentes como escolha dos elementos de interação a serem usados (janelas,menus, etc.) e escolha da apresentação destes elementos em momentos (e níveis) diferenciados. /CAR 83/ observa que o projeto de IUs é mais fácil se claramente organizado em níveis de complexidade.

- Para o programador de aplicação: poder usar os elementos da IU de modo mais abstrato, sem se preocupar em programar todos os passos para exibí-los/manipulá-los. A intenção é implementar, sobre as funções primitivas das FIUs comerciais, novos níveis de abstrações mais próximos das abstrações usadas ao nível das aplicações. Estas novas abstrações trabalhariam geralmente com elementos como menus, formulários de entrada ("form-fill"), ícones, janelas, etc.

Ao projetista de interface são necessárias abstrações que permitam especificar um tipo de informação de cada vez: primeiro, que objetos serão usados na IU e, depois, suas apresentações.

Ao programador de aplicação restaria apenas saber o quê fazer com relação à IU (sua seqüenciação ou controle), não precisando saber como fazê-lo. Por exemplo, para exibir uma janela, o programador de aplicação apenas forneceria o tipo de janela e as coordenadas, não precisando programar os passos necessários como abrir uma "viewport", salvar conteúdos da porção da tela sobreposta pela janela, traçar as linhas da estrutura da janela, etc.

Para estabelecer, para a FIU Canônica, um conjunto de objetos de IU de uso genérico pelas aplicações, decidiu-se selecionar, a partir dos elementos/operações disponíveis nas FIUs apresentadas, um conjunto "default" de objetos suportados e dar possibilidade de compor novos objetos a partir deles. Este conjunto tem uma arquitetura aberta, que permite a substituição/extensão de objetos ou a redefinição dos papéis de determinados objetos em cada IU definida.

Como a FIU Canônica propicia portabilidade?

Uma solução para permitir a portabilidade de programas com relação às FIUs é a definição de uma camada padronizada entre a FIU e a aplicação. A FIU Canônica é esta camada padronizada. Com o estabelecimento desta padronização, possibilita-se que todas as interfaces e suas comunicações com as aplicações obedeçam ao **mesmo protocolo.** Aspectos permitidos pela FIU Canônica mas não implementados nas FIUs reais poderão estar presentes em novas versões destas. A idéia inicial desta padronização é devida ao prof. Roberto Tom Price, no âmbito do projeto AMADEUS /PRI 88/ e já apresentada em /FPP 90/.

A figura 4.3 mostra uma aplicação desenvolvida para diferentes ambientes utilizando a FIU Canônica. O uso de um protocolo de diálogo interno padronizado possibilita que a aplicação seja mais portável.

A partir da criação da FIU canônica, os programas são escritos em função desta, devendo o uso de objetos e/ou rotinas de interação ser mapeado para o uso do conjunto

(possivelmente unitário) dos correspondentes objetos/rotinas primitivas da FIU sobre a qual o programa será executado. Os programas, desta forma, tornam-se portáveis, uma vez que são não apenas independentes de dispositivos (vídeo1, vídeo2, etc.) e de meios (vídeo, "plotter", etc.) mas possivelmente também de FIUs. A FIU Canônica corresponde ao conceito de "socket" de diálogo proposto por /COU 85/.

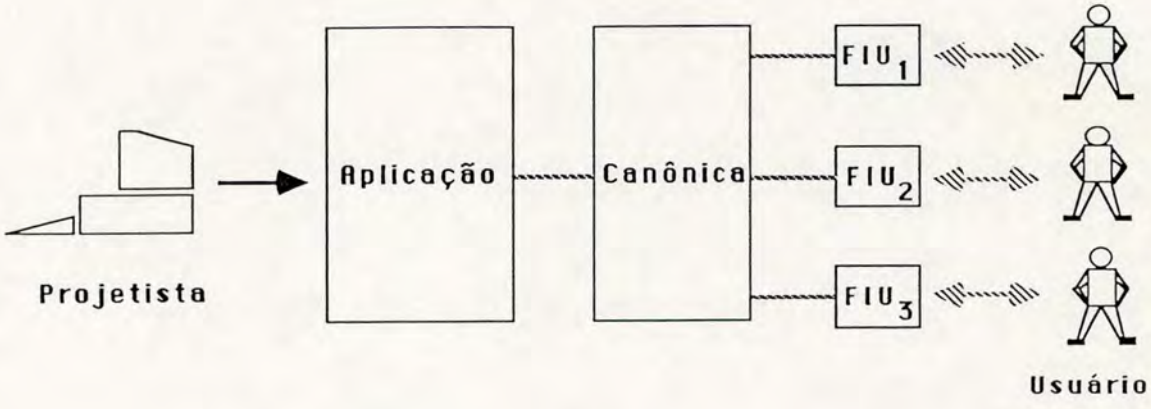

Figura 4.3 - Uma aplicação escrita para diferentes ambientes usando a FIU Canônica

Em verdade, a maioria das aplicações pode ser portada entre ambientes heterogêneos usando a FIU Canônica se:

- o código da aplicação está escrito de forma padronizada (sem características específicas de opções de compiladores, por exemplo) numa linguagem presente nos vários ambientes;

- usam os objetos de IU da FIU Canônica e
- evitam o uso de chamadas às FIUs ou sistemas operacionais subjacentes e o acesso direto a dispositivos físicos de E/S.

4.3 Requisitos adicionais da FIU Canônica

Para cobrir uma grande variedade de aplicações e grupos de usuários, uma FIU deve suportar uma variedade de estilos de diálogo. Alguns dos estilos que podem ser suportados por uma FIU são /MYE 89/:

- linguagem de comandos;
- formulários ("form-fill") e menus;
- perguntas/respostas ("question/answer");
- Manipulação gráfica com feedback léxico;
- Manipulação gráfica com feedback semântico;
- Manipulação direta de imagens gráficas;

As três primeiras correspondem à metáfora conversacional e as três últimas à metáfora espacial. A FIU Canônica deve lidar naturalmente com estes estilos e metáforas.

FIUs devem ser **extensíveis** porque sua funcionalidade nunca é absolutamente completa (os tipos de IU que permite são claramente limitados). Uma vez que as possibilidades para interfaces homem-computador são

ilimitadas, FIUs específicas não podem atender a todas necessidades. Há no mínimo duas maneiras de tornar as FIUs extensíveis /HAH 89/ : modificar as FIUs ou modificar as especificações (representações) de IUs produzidas pelas FIUs. Uma FIU deve ser um sistema aberto, o que significa que suas características (objetos e operações) podem ser facilmente estendidas. A capacidade de extensão torna a FIU mais genérica.

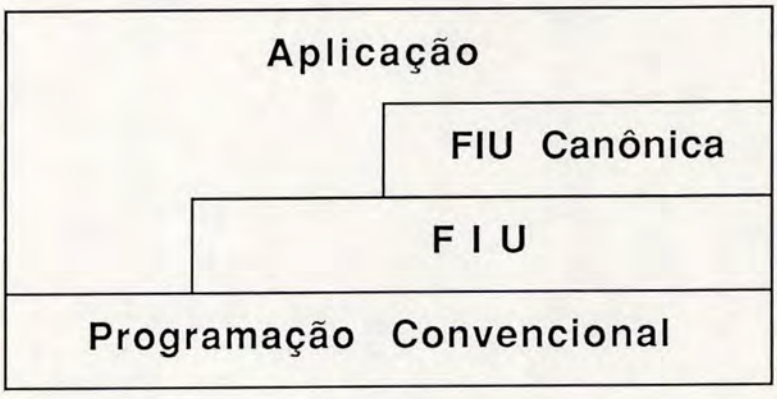

Figura 4.4 - Possibilidade de escape da FIU Canônica

Nos casos em que a FIU é inadequada e extensão não é viável ou factível (como, por exemplo, uma característica rara ou um dispositivo de E/S não convencional) deve ser possível "**escapar**" da FIU e usar métodos de programação convencionais para implementar o desejado. O uso da FIU Canônica deve portanto ser compatível com o uso de rotinas da FIU específica assim como de programação convencional no mesmo ambiente, conforme figura 4.4 . Sem esta habilidade de "escapar" da FIU, as limitações inevitáveis tornam-se o ponto final para os usuários projetistas.

A FIU Canônica portanto tem as seguintes peculiaridades:

a) é constituída de primitivas que implementam abstrações que isolam o programador de aplicação dos detalhes relativos a dispositivos de E/S físicos e relativos a decisões de projeto/especificação de IU (encargo do projetista de interface);

b) possibilita níveis apropriados para a especificação da IU pelo projetista de interface;

c) suporta prototipação, permitindo fácil e rápida definição de objetos de IU;

d) suporta diferentes conjuntos de objetos e técnicas de interação, cada qual podendo ser associado a estilos de IU, "feedback" semântico e valores "default" diferentes;

e) o protocolo entre a FIU Canônica e a aplicação é um protocolo orientado a objetos, de alto nível, proporcionando abstrações adequadas para o uso do programador da aplicação; e

f) possibilita a portabilidade de programas interativos.

5 ORIENTAÇÃO A OBJETOS E INTERFACE COM USUÁRIO: UMA APROXIMAÇÃO NATURAL

"Quem pisou primeiro neste barro?
E como esta lama pôde ficar tão deliciosa?"
John Cage

Neste capítulo serão apresentados alguns conceitos e definições do PDOO, visando uma familiaridade com suas concepções. As razões do uso do PDOO em IU e da escolha da abordagem de PDOO utilizada são também discutidas.

5.1 PDOO : Conceitos Básicos

Na modelagem segundo o paradigma computacional convencional (representado pelas linguagens ditas **algorítmicas**) diferencia-se dados (operandos) e instruções (operadores), com a ação concentrando-se na seqüência de instruções enquanto os dados são passivos, atuando como parâmetros/argumentos das instruções.

A modelagem com o Paradigma de Desenvolvimento Orientado a Objetos (PDOO) distingue-se fundamentalmente da anterior por não enfatizar esta distinção entre dados e instruções /COX 84/.

Quando se usa o PDOO, no entanto, não há consenso em relação a um conjunto padrão de conceitos e implementações. Os conceitos adotados neste trabalho são baseados fortemente nos propostos por /GIR 90/ e /MEY 88/.

A noção central do PDOO é a noção de **objeto.** Objeto é uma entidade com capacidade de armazenar informação e manipulá-la ou, em outras palavras, um encapsulamento de uma estrutura de dados junto com as operações para alterá-la ou fazê-la exibir comportamento /GIR 90/. Estas operações são denominadas **métodos.**

A única maneira de interagir com um objeto é enviando-lhe uma **mensagem.** Uma mensagem é uma requisição a um objeto receptor para que execute um de seus métodos. Este mecanismo fornece também uma separação entre a **implementação** de um objeto (seu conjunto de atributos e o código de seus métodos) e seu **protocolo,** que diz respeito ao seu comportamento externo. Os métodos são as rotinas que implementam a funcionalidade de cada mensagem. Em Eiffel /MEY 88/ , um atributo pode fazer parte também do protocolo do objeto, podendo ter seu valor consultado/atualizado. Com isto, evita-se a definição de mensagens apenas para consultar/modificar valores de atributos. Cada objeto, então, reage às mensagens que recebe e que constituem o seu protocolo de comunicação, rejeitando as demais.

Na grande maioria das Linguagens Orientadas a Objeto (LOO), os objetos são agrupados em **classes.** Objetos de uma mesma classe (denominados **instâncias** desta classe) possuem a mesma estrutura de dados interna (denominados **atributos** ou variáveis de instância) e os mesmos métodos.

Os objetos de uma classe (instâncias) diferenciam-se entre si pelo seu estado, isto é, pelos valores de seus atributos. Atributos são as variáveis que formam as estruturas de dados do objeto. Seus valores podem

expressar o estado do objeto ou referenciar outros objetos /SIB 86/.

Classes podem ser reutilizadas de 2 maneiras. A primeira é simplesmente a instanciação de determinadas classes, produzindo novos objetos para o sistema. A segunda consiste no aproveitamento de classes para a construção de novas classes. Isto é feito utilizando-se instâncias de classes determinadas na descrição de novas classes (composição) ou através do mecanismo de herança.

O conceito de **herança**, em LOO, associa-se à idéia de que, na definição de uma nova classe, é conveniente partir de atributos e métodos de uma outra classe já existente e adicionar algo mais para obter a nova classe. Diz-se que a nova classe herdou os atributos e os métodos a partir da antiga. A herança é um mecanismo muito importante e útil, pois incorpora facilidades de compartilhamento de código e reusabilidade às classes.

O mecanismo que permite a uma nova classe herdar variáveis e métodos de mais de uma classe é conhecido por herança múltipla.

Uma **LOO pura** segue o projeto proposto pelo sistema Smalltalk /GOL 83c/. Nestas linguagens, só existem os conceitos próprios da estruturação por objetos: objetos, classes e mensagens. Conseqüentemente, um conjunto de classes originais (pré-existentes) armazenadas numa biblioteca devem proporcionar a funcionalidade mínima da linguagem.

A linguagem **Eiffel** é uma linguagem orientada a objetos pura mas que permite acoplamento com outras linguagens ditas convencionais. As principais diferenças entre Eiffel e Smalltalk são:

a) Smalltalk não diferencia classes e objetos. Tudo é objeto: uma classe é instância de uma classe de mais alto nível, a metaclasse. Eiffel faz uma clara distinção: classe é a definição de objetos (existe a tempo de compilação); objetos (instâncias da classe) existem a tempo de execução;

b) Objetos em Eiffel são tipados (amarração estática), em contraste com Smalltalk onde há amarração dinâmica de um objeto a um tipo;

c) Em Smalltalk, apenas métodos podem ser acessíveis externamente via mensagens; em Eiffel, tanto atributos quanto métodos (**rotinas**, na nomenclatura Eiffel) fazem parte do protocolo;

LOO híbridas são extensões de linguagens convencionais (Pascal, C, Lisp) que incorporam noções de representação por objetos.

5.2 PDOO e IUs: a aproximação

Assim como os usuários esperam consistência entre as aplicações no que diz respeito às facilidades de interação da aplicação (e no tratamento semelhante de objetos semelhantes em aplicações diferentes), os usuários projetistas precisam de um suporte consistente para implementar este tratamento semelhante nas variadas aplicações /BEN 86/.

IUs são inerentemente difíceis de construir sem abstrações que simplifiquem este processo de implementação.

As características do PDOO, vistas na seção 5.1 - encapsulamento (abstração de dados e "information hiding"), herança de atributos e métodos, trocas de mensagens - tornam-no uma excelente plataforma para o desenvolvimento de IUs /HAR 89/.

O próprio uso do PDOO está intimamente ligado a um novo conceito de IU: a manipulação direta com janelas, ícones e menus facilitando a atividade de usuários e projetistas surgiu junto com o ambiente orientado a objetos Smalltalk /GOL 83a/.

Entre as principais vantagens do uso do PDOO para IU tem-se /SIB 86/, /HAR 89/:

i) PDOO provê um mecanismo natural para representar a IU em vários níveis de abstração em suas hierarquias de classes, onde um objeto pode ser tudo desde um conceito de alto nível, como um editor de ícones, até um conceito muito baixo como um controlador de dispositivo de E/S. Estes diferentes níveis de abstração facilitam a representação natural de conceitos em cada nível da IU;

ii) Facilidade de combinar objetos simples em objetos mais complexos, tornando possível definir novos componentes de um sistema, até interativamente, a partir dos já existentes;

iii) Flexibilidade inerente aos sistemas definidos em termos de protocolos de mensagens: fica fácil adicionar novas capacidades ao sistema sem grande necessidade de recodificação;

iv) Reusabilidade, provendo um conjunto de objetos que podem ser usados como "blocos de construção" pelos desenvolvedores, encurtando o ciclo (tempo) de desenvolvimento e evitando a reimplementação de funções de IU. Novas IUs podem ser criadas herdando características de IUs já existentes.

v) separação da representação e implementação dos objetos de IU;

vi) menor volume de código em comparação com o modelo algorítmico tradicional.

As principais desvantagens do uso do PDOO para IUs incluem:

i) curva de aprendizado íngreme para programadores não acostumados a PDOO;

ii) penalidade de performance devido à amarração dinâmica e troca de mensagens (que são mais custosas que simples chamadas de rotinas), se o sistema orientado a objetos é interpretado;

iii) dificuldades de integração de IUs orientadas a objeto com software já existente desenvolvido através de paradigmas ou metodologias convencionais; isto inibe que orientação a objetos seja usada apenas para IUs, já que o resto do sistema é projetado segundo outras metodologias.

5.3 Enfoque de PDOO adotado

Os conceitos (e a sintaxe) de orientação a objetos usados na dissertação são os conceitos difundidos para PDOO. Para a definição do modelo representacional da FIU Canônica três enfoques poderiam ser adotados:

1-Uso de LOO puras, como Smalltalk e Eiffel;

2-Uso de LOO híbridas (em geral, extensões de C, Pascal ou Lisp);

3-Criação de uma nova linguagem orientada a objetos específica para definição de IUs.

O terceiro enfoque, a criação de uma YAOOL ("Yet Another Object Oriented Language"), está fora do escopo deste trabalho, além de contrariar um princípio básico do PDOO: a reusabilidade.

O segundo enfoque permite integração com código escrito em outra linguagem - um importante aspecto para alcançar reusabilidade - mas o uso de apenas uma linguagem (a convencional que foi estendida) é muito restritivo para a proposta da FIU Canônica.

No primeiro enfoque, o uso de Smalltalk, apesar da pureza conceitual, apresentava dois problemas:

- dificultava a integração com componentes escritos em outras linguagens, requisito necessário para IUs de aplicações que não são orientadas a objetos;

- Smalltalk não é apenas uma linguagem mas um ambiente de programação /GOL 83a/.

O enfoque adotado para a notação do Canonicus é baseado na linguagem Eiffel. Eiffel é uma LOO pura, portanto mais homogênea que as LOO híbridas, mas com uma implementação aberta, permitindo integração com códigos escritos em várias outras linguagens /MEY 88/.

Das desvantagens do PDOO apresentadas, Eiffel resolve as duas últimas: não possui os problemas de performance da amarração dinâmica e permite fácil integração com programas (e linguagens) desenvolvidos com outras metodologias. A curva de aprendizado parece inevitável e deve ser enfrentada por qualquer nova tecnologia.

6 A FIU CANÔNICA

6.1 Introdução

Para ajudar os usuários projetistas a criar IUs, devem ser consideradas as seguintes questões /LIN 89/:

a) tipos de IUs suportadas;

b) tipos de controle de interação ou forma de diálogo interno;

c) qual conjunto de abstrações é escolhido para construção de IUs, o que se reflete em como os componentes da IU são estruturados e como eles interagem entre si; e

d) como o usuário projetista constrói uma IU usando estas abstrações.

As três primeiras questões são resolvidas pela arquitetura da FIU utilizada, enquanto a última é resolvida pelo seu MRIU /HAH 89/.

Neste capítulo é apresentada a arquitetura do tipo "toolbox" da FIU Canônica e seus componentes principais, os objetos de interface com o usuário. Uma explicação de sua composição e de como eles podem ser organizados em níveis é também fornecida. Além disto, as alterações na arquitetura da FIU Canônica necessárias para a transformar em uma "toolkit" são sucintamente descritas.

O MRIU da FIU Canônica é apresentado no capítulo 7.

6.2 Arquitetura da FIU Canônica

A FIU Canônica deve suportar os diferentes tipos de metáforas (conversacional e espacial), os diferentes estilos de diálogo (ver seção 4.3 no capítulo 4) e diferentes apresentações de IUs. Esta flexibilidade é usualmente provida pelas arquiteturas de FIUs através da divisão da IU em componentes funcionais.

A arquitetura da FIU Canônica é uma arquitetura orientada a objetos composta de um conjunto de **objetos de interface com o usuário**, ou abreviadamente, **objetos de IU.**

Como visto no capítulo 5, objetos são naturais para representar os elementos de uma IU. Objetos provêem um bom mecanismo de abstração, encapsulando estruturas e operações e com a herança fazendo a extensão mais fácil. /LIN 89/ mostra que, comparada com uma implementação procedural, IUs escritas em uma LOO são significativamente mais fáceis de desenvolver e manter.

Por isto, para a definição da FIU Canônica foi adotado o **PDOO**, que é um paradigma de especificação e implementação que:

a) possui a capacidade de lidar com níveis de abstrações;

b) é completo para especificação e projeto de IUs (aspectos estáticos e dinâmicos);

c) é flexível, permitindo evolução e modificação;

d) permite reusabilidade.

Lato sensu, o termo "objeto de IU" serve para tornar clara a separação da IU da aplicação. Os objetos de uma IU são representações de elementos da aplicação, refletindo suas alterações de estado. Por outro lado, objetos de IU são também a maneira pela qual o usuário interage com a aplicação. Há, desta forma, dois agentes que executam ações sobre objetos da IU: a aplicação e o usuário.

Segundo /HUD 87/, para suportar as ações destes agentes sobre os objetos de IU, uma FIU baseada em **objetos compartilhados** é preferível a FIUs convencionais baseadas em rotinas de ação semântica. Ao contrário de outras arquiteturas orientadas a objetos em que alguns objetos são manipulados pelo usuário, outros pela aplicação e outros são intermediários (como por exemplo /SIB 86/), numa arquitetura de objetos compartilhados os objetos são manipulados tanto pelo usuário quanto pela aplicação. Exemplos de sistemas com esta arquitetura são Higgens /HUD 86/ e o GROW /BAR 86/.

Objetos de IU são objetos compartilhados pelos agentes - usuário e aplicação, conforme ilustra a figura 6.1.

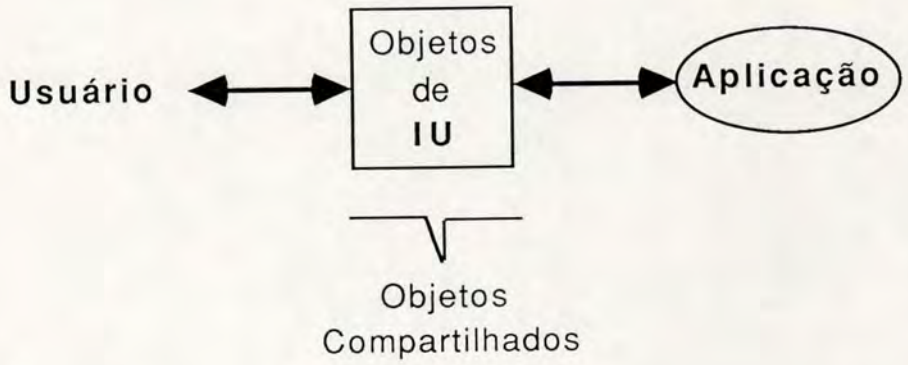

Figura 6.1 - Objetos de IU acessíveis à aplicação e ao usuário

Quando mudanças nos objetos são feitas por uma parte, a outra é notificada. Isto permite que o "feedback" semântico seja manipulado pela FIU provendo, assim, mais suporte automatizado à tarefa de projeto e implementação de "boas" IUs /LIE 86/.

Um modo efetivo de suportar objetos de IU é uma **"toolbox" de objetos de IU** primitivos que possuem um protocolo de mensagens para definir seu comportamento. Estes objetos primitivos podem ser combinados para formar objetos mais complexos ou especializados para diferentes tipos de uso. Uma "toolbox" é um mecanismo simples porém flexível, servindo para especificar diversas IUs, com graus variados de sofisticação. Esta concepção (objetos primitivos com facilidades de composição) facilita a construção da "toolbox" e as tarefas do projetista de interfaces. A construção da "toolbox" concentra-se assim em implementar isoladamente objetos primitivos para que o projetista de interface esteja livre para combiná-los de modo a compor IUs mais adequadas à aplicação.

Há duas classes principais de objetos de IU: manipuladores de eventos e objetos de interação, que serão descritos adiante.

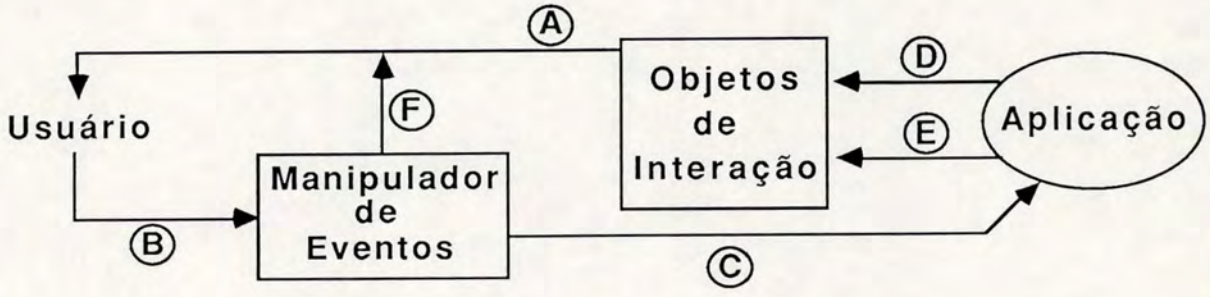

Figura 6.2 - Arquitetura da "toolbox" Canônica

A arquitetura do tipo "toolbox" proposta para a FIU Canônica é apresentada na figura 6.2. As letras de A a F correspondem a comunicações entre os agentes e objetos de IU. Esta arquitetura apresenta um comportamento descrito pelos seguintes passos:

1) Os objetos de IU (definidos através de um MRIU pelo projetista de interface) são apresentados ao usuário (comunicação A);

2) O usuário manipula os objetos através de ações (comunicação B) ; durante a ação do usuário é fornecido o "feedback" dinâmico (comunicação F) para eventos e movimentos do cursor do teclado e "mouse".

3) As ações são interpretadas pelo manipulador de eventos que gera eventos compreensíveis pela aplicação (comunicação C);

4) Como o controle, usando-se uma "toolbox", é interno, a aplicação envia mensagens aos objetos de interação e ao manipuladores de evento para chamar rotinas de interação, quando necessários para sua computação (comunicação D).

5) Os resultados a serem apresentados ("feedback bruto") são enviados para os objetos de interação (comunicação E);

6) Retorno ao passo 1, sendo que a modificação da apresentação do objeto de interação é o "feedback" decorado, em resposta à ação do usuário (comunicação A).

Note-se que, na arquitetura da "toolbox" Canônica, existem as seguintes comunicações na descrição do diálogo interno:

Comunicação C = eventos (rotinas do manipulador de eventos chamadas diretamente de dentro da aplicação);

Comunicação D = mensagens aos objetos de interação para chamar rotinas para manipulação dos objetos de interação e alterar atributos dos objetos;

Comunicação E = dados de resultado/retorno para exibição;

6.3 Objetos de Interface com o Usuário

Os objetos de IU são os mecanismos pelos quais uma aplicação pode adquirir/exibir informações e diretivas (instruções, ações, comandos) do/ao usuário. Uma IU é então um conjunto de objetos de IU que podem ser usados diretamente, especializados para uma ambiente de interação específico (Mac ou MS-Windows) ou para uma determinada aplicação.

Geralmente, o critério de classificação de objetos de IU em hierarquias é baseado em facilidade de implementação. Uma classe **caixa de diálogo** é vista, por exemplo, como uma subclasse da classe **janela** pois é considerada um caso particular de janela, utilizando suas rotinas. O critério adotado para a FIU Canônica é a **facilidade de uso** para os usuários projetistas, mesmo que sejam mais difíceis de implementar. Então, **caixa de diálogo** e **janela** são classes distintas.

Como visto, há duas classes principais de objetos de IU:

a) Manipuladores de eventos e

b) Objetos de interação.

6.3.1 Manipuladores de Eventos

A classe de objetos **Manipuladores de Eventos** (também chamados Controladores de Entrada /GOL 83b/) tem por objetivo entender e verificar a entrada do usuário. O domínio de "entrada do usuário" inclui reconhecer e fornecer "feedback" dinâmico para os eventos e movimentos do mouse, cursor do teclado ou outro dispositivo de entrada, seja ele físico ou lógico /COU 85, ARS 82/.

É uma classe peculiar pois possui uma única instância. Toda entrada do usuário é isolada no Manipulador de Eventos, tornando mais flexível adicionar tratamento de novos dispositivos sem alterar os objetos de interação ou a aplicação.

O Manipulador de Eventos reconhece as ações do usuário coordenando a ação dos dispositivos de entrada ("mouse" e teclado, etc.), gerando mensagens de eventos. Além disso, é encarregado também de prover "feedback" dinâmico para os eventos e movimentos dos dispositivos de entrada como, por exemplo, movimentação do cursor ("mouse" ou teclado), um "click" de algum botão do "mouse" ou o apertar de uma tecla.

Cada evento na FIU Canônica é relacionado a um **tipo.** O número de tipos de eventos da FIU Canônica representa um meio termo entre os 11 tipos de eventos do Mac e os 96 tipos de mensagens do MS-W. Há eventos para os dispositivos normais como "mouse" e teclado. Alguns eventos específicos já são determinados, ou seja, as tarefas a serem realizadas já são previstas (como, por exemplo ATIVAR_JANELA), criando um suporte de mais alto nível a

eventos deste tipo. O usuário projetista pode no entanto usar os eventos simples como controle de "click" de "mouse" e teclas digitadas, se quiser. Os tipos de eventos do usuário previstos na FIU Canônica e mais informações sobre os manipuladores de eventos encontram-se no Anexo C.

Na "toolbox" Canônica, a aplicação chama diretamente as rotinas (mensagens) do manipulador de eventos.

6.3.2 Objetos de Interação

O conceito de um **objeto de interação** (também chamado Objeto Gráfico /BAR 86/) é central a FIU Canônica.

Objetos de interação são usados como blocos de construção para desenvolver IUs. Um objeto de interação, por definição, é todo elemento visualmente distinto de uma tela. Uma lista corrente de objetos de interação inclui janelas, menus, caixas de diálogo, ícones, textos fixos, textos editáveis, alertas e controles (botões, "radio buttons", "check boxes", "dials").

Nesta lista constam apenas objetos relativos à IU. As FIUs comerciais apresentadas contém também elementos e rotinas que não dizem respeito a IU e que foram nelas incluídos para fornecer serviços secundários, como por exemplo gerência de memória, comunicação de dados, desenho (operações sobre gráficos complexos) e concorrência. Estes serviços não são considerados pertinentes à IU e não foram, assim como rotinas utilitárias e/ou de baixo nível

(acessíveis através de rotinas de nível mais alto), previstos na FIU Canônica.

Os objetos de interação são arranjados em hierarquias tal que procedimentos e dados podem ser compartilhados através de **herança**. Esta característica facilita a especialização e a reusabilidade e pode ser achada em outros sistemas /GOL 83b, SCH 86, BAR 86/.

A descrição de um objeto de interação inclui a descrição dos seus atributos e operações possíveis sobre este objeto.

Atributos são as propriedades do objeto como um todo. Incluem valores pertencentes a tipos primitivos (inteiro, "string", etc) ou identificadores de outros objetos de interação. Atributos podem ser acessados via mensagens se incluídos no protocolo do objeto de interação.

Operações são os procedimentos que podem ser feitos sobre um objeto de interação cujo resultado pode ser a mudança de seu estado (valores de atributos), o envio de mensagens a outros objetos (ou outros agentes do sistema), a resposta a uma mensagem recebida, a criação de novos objetos ou a combinação destas ações /GIR 90/ /DAN 88/. Uma operação é solicitada na forma de uma mensagem que o objeto entende (que faz parte de seu protocolo de comunicação) e é a única maneira de um objeto de interação ser manipulado.

6.3.3 Níveis de definição de objetos

A necessidade de separar papéis de programador de aplicação e projetista de interface leva à definição de dois níveis de definição de objetos: o nível em que é descrita a composição e função dos objetos de interação, denominado **nível abstrato** ou **genérico;** e o nível que descreve a sua aparência (apresentação), denominado **nível concreto.**

A combinação dos atributos e operações descrevem um objeto no nível abstrato, o **objeto genérico.** O objeto genérico não é uma instância da classe genérica mas sim uma descrição abstrata da composição e função de um objeto de IU. A descrição da aparência do objeto de IU (sua apresentação) é especificada pelos atributos físicos (concretos) disponíveis na forma concreta escolhida, ou seja, na FIU específica utilizada. A apresentação de um objeto genérico é especificada em outro tipo de definição, que mapeia entre os atributos concretos (de apresentação) do objeto genérico e uma coleção de características de apresentação específicas de alguma FIU, usadas para representar instâncias de objetos genéricos. Para a FIU Canônica as classes concretas são subclasses das classes definidoras dos objetos genéricos.

Cada objeto de IU é, em verdade, definido como uma instância de uma hierarquia de classes:

-**Classes genéricas**, onde são definidos os objetos genéricos e

-**Classes concretas,** subclasses das classes genéricas, onde estão as instâncias de objetos de IU.

O projetista de interface define os objetos (sua propriedades e seus relacionamentos) e seus operadores no nível abstrato. A seguir define, no nível concreto, a apresentação dos objetos usados no nível abstrato e os detalhes para sua apresentação sobre uma FIU específica.

O **programador de aplicação usa atributos e operações existentes no nível abstrato,** ficando transparente para ele os detalhes presentes no nível concreto.

Uma classe genérica possui todos as descrições independentes de apresentação e de FIU. É a abstração das propriedades (atributos) e comportamentos (métodos) dos objetos de IU das FIU específicas estendida com atributos e métodos considerados necessários à definição de IUs mais completas.

Uma classe genérica pode ter várias subclasses concretas significando que é possível haver diferentes níveis concretos para um mesmo nível abstrato de um objeto.

Uma janela pode ter bordas simples, cantos arredondados,barra de títulos escura e uma "close box" no extremo esquerdo da barra de títulos (janela tipo **rDocProc** do Mac Toolbox), como pode ter bordas simples, cantos em ângulo reto, barra de títulos escura, uma área de ícones na área inferior da tela, uma caixa representando o menu do

sistema no extremo esquerdo da barra de título (janela de **aplicação** do MS-Windows). Ambas são aparências diferentes (duas ocorrências no nível concreto) do mesmo **objeto genérico janela.**

A descrição no nível abstrato é essencialmente um conjunto de descrições parametrizáveis de objetos de IU, que serão manipulados pela aplicação de forma **independente de FIU.**

A descrição no nível concreto é uma representação do objeto em uma forma apropriada para exibição e manipulação pelo usuário, **segundo uma FIU específica.**

Uma instância de objeto de interação é uma instância de uma classe concreta (descrita no nível concreto), o qual é uma das aparências possíveis (uma subclasse) da classe genérica ou abstrata (descrita no nível abstrato), conforme figura 6.3.

Uma peculiaridade da FIU Canônica é o modo de definir as operações sobre um objeto de interação genérico (no nível abstrato). Os métodos das classes genéricas são **métodos deferidos**, ou seja, métodos definidos na classe mas realmente implementados nas suas subclasses. Este conceito existe na linguagem Eiffel /MEY 88/. Métodos deferidos podem ser encarados como a descrição genérica de um grupo de implementações específicas. Cada maneira diferente de implementar o método genérico é uma das implementações específicas. Não há definição efetiva de um método na classe genérica, apenas a especificação na forma de um método deferido. Por isto, as classes genéricas são também

chamadas **classes deferidas** e não podem ter instâncias, pois não há métodos da classe para manipulá-las.

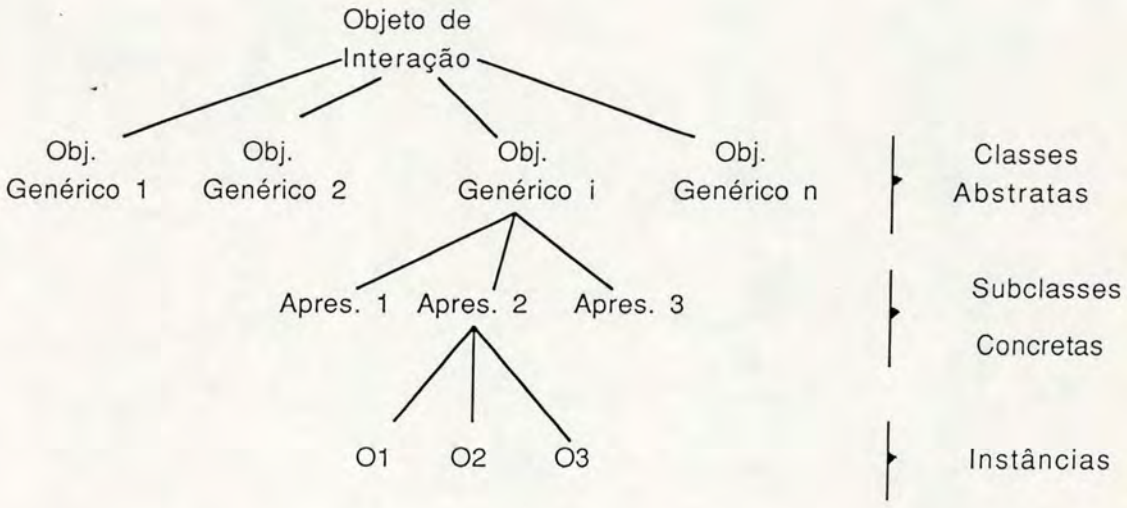

Figura 6.3 - Hierarquia de classes de objetos de IU

Em resumo, uma classe genérica possui a descrição dos objetos genéricos e os métodos que operam sobre eles. As subclasses concretas possuem a definição dos atributos de apresentação dos objetos e a implementação dos métodos descritos nas classes genéricas.

Métodos deferidos são particularmente úteis para uso de operações (mensagens) sobre os objetos de interação por parte da aplicação. A aplicação deve usar os métodos num nível abstrato, que escondem do programador de aplicação os detalhes de manipulação e exibição dos objetos. Estes métodos podem corresponder a várias versões de implementações diferentes (dependendo de qual subclasse

concreta é pai da instância) e não se precisa tomar conhecimento de qual implementação está sendo realmente executada.

Em verdade, cada implementação de um método deferido é a chamada da(s) rotina(s) da FIU subjacente que realiza(m) a função definida por este método.

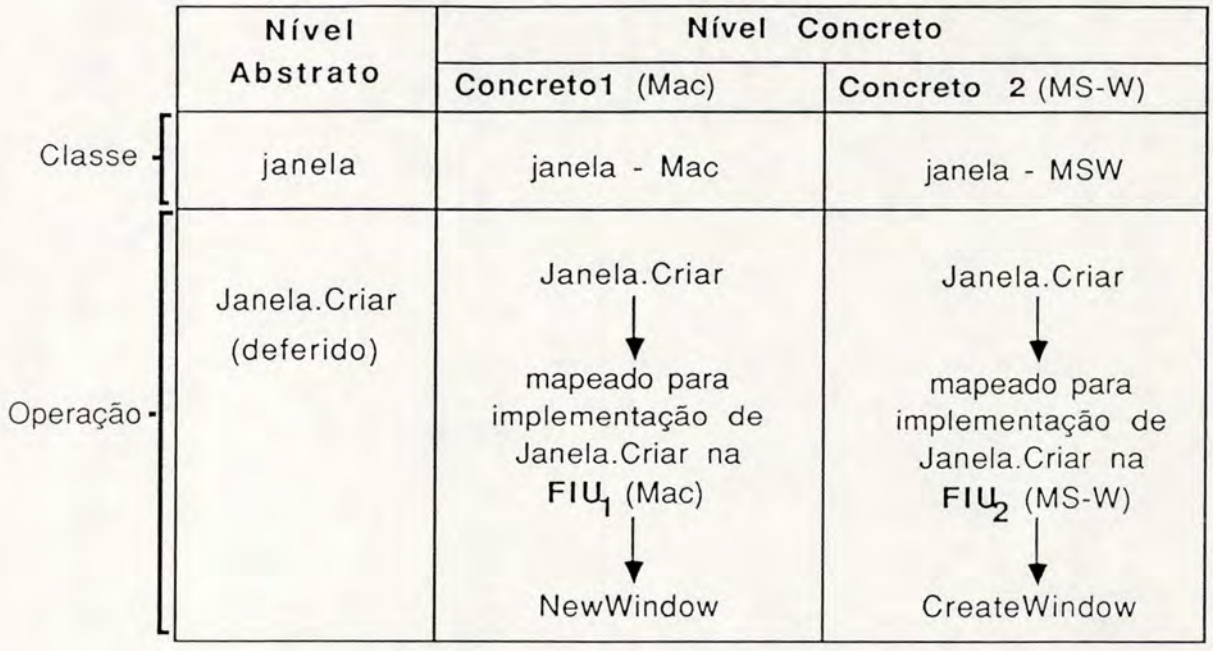

Figura 6.4 - Implementação de métodos das classes abstratas: um exemplo

Versões efetivas do método deferido são dadas nas subclasses da classe genérica, ou seja, nas classes concretas. Uma classe concreta é então onde são descritos os elementos e métodos de uma FIU específica e uma classe genérica é a definição abstrata (canônica e virtual) dos objetos e métodos deferidos. Isto é ilustrado na figura 6.4.

Uma classe genérica é a representação abstrata de um grupo de implementações de suas subclasses concretas, espelhando o fato de que a FIU Canônica é a abstração das FIUs em que se baseia.

O mecanismo básico para a definição de objetos genéricos é a **abstração.** A pesquisa em IA e em BD identifica tipos fundamentais de abstrações (cada qual com um tipo inverso correspondente), entre as quais incluem-se generalização (inversa:especialização) e agregação (inversa: decomposição) /SMI 77/.

Estas duas abstrações apresentam a propriedade de serem **ortogonais** entre si. Na FIU Canônica, utiliza-se estas abstrações em duas hierarquias ortogonais de objetos:

- hierarquia **taxonômica,** com generalizações/especializações envolvendo classes genéricas, suas sub-classes concretas e instâncias de objetos e
- hierarquia **composicional,** com agregação/decomposição envolvendo objetos compostos (painéis) e objetos componentes.

A herança de propriedades (atributos, métodos) fica implícita na hierarquia taxonômica. Classes são inseridas em uma hierarquia de forma que uma classe mais especializada (subclasse) herda todas as propriedades da classe mais geral (superclasse) à qual se subordina na hierarquia. A uma subclasse podem ser agregados novos atributos e/ou métodos que detalham ainda mais a estrutura e o comportamento das instâncias derivadas das subclasses. A adição de um novo método ou atributo na especificação de uma subclasse eventualmente mascara outro de mesmo nome

presente nas suas superclasses (diretas ou indiretas) já que em caso de ambigüidade aplica-se o mais específico.

Na hierarquia composicional utiliza-se referências a outras classes (as dos objetos componentes) na definição de atributos de novas classes. Estas referências, junto com outros atributos, constituem a estrutura (conjunto de atributos) do objeto da nova classe. Diz-se então que um objeto da nova classe é composto de objetos componentes. Nesta hierarquia não existe o mecanismo de herança.

6.3.4 Painéis

Como visto, além do relacionamento de herança, objetos de interação podem se relacionar através de relacionamentos de composição. Composição permite que várias classes possam ser combinadas formando uma classe composta, denominada **classe painel.** Uma instância de uma classe painel é denominada **objeto composto** ou simplesmente **painel,** e pode ser manipulada como uma unidade.

Objetos compostos são ligados a seus componentes através de atributos contendo uma referência ao componente. Componentes podem por sua vez ser objetos compostos de modo que a estrutura composta pode ter uma profundidade arbitrária. Cada um dos componentes é ele próprio um objeto com sua própria descrição.

Um objeto composto (painel) é resultado da agregação de vários objetos componentes e pode ser tratado em unidade como um único objeto.

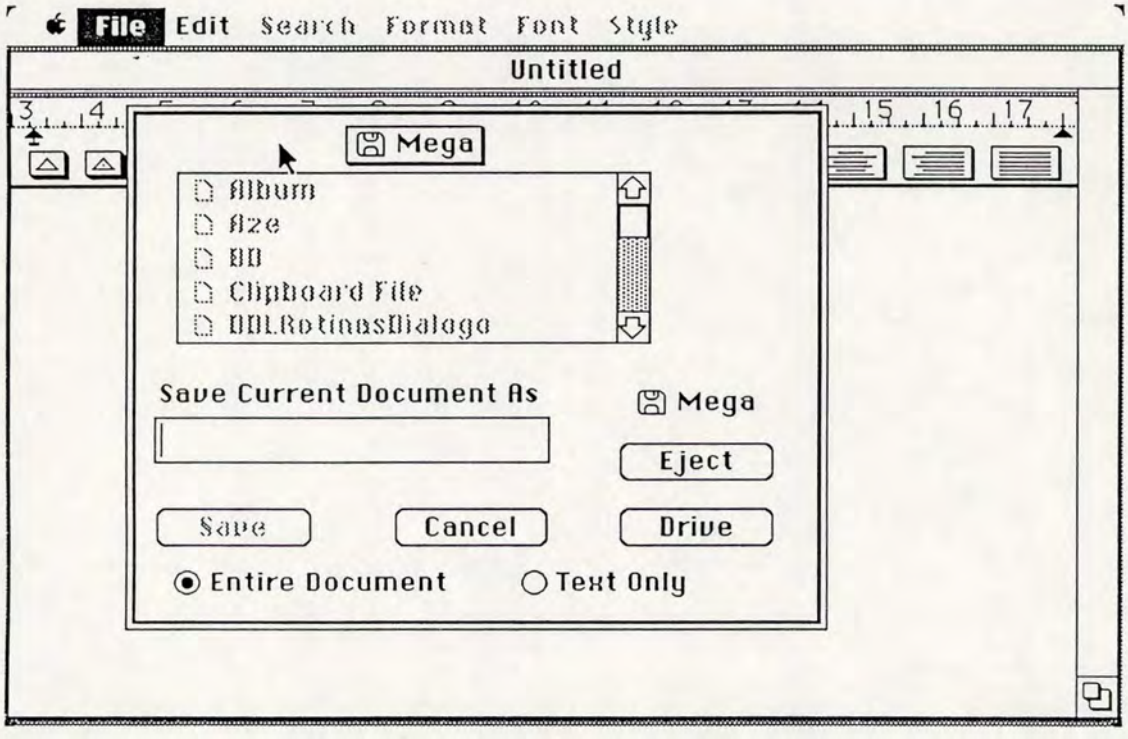

Figura 6.5 - Exemplo de painel

Um exemplo típico de painel é um objeto composto de (ver figura 6.5):

- duas janelas (uma principal e uma de diálogo);
- uma caixa de diálogo;
- uma barra de menus e menus cortinas associados (símbolo de maçã, "File", "Edit", "Search", "Format", "Font", "Style");

- controles (Botões: "Save", "Cancel", "Eject", "Drive"; Radio Buttons: "Entire Document" e "Text Only").

Muitas operações sobre o objeto composto, como "zoom", movimentação e remoção, são aplicadas também a seus componentes.

Operações gráficas, como "mover" e "esconder", podem ser aplicadas ao objeto composto como um todo ou individualmente aos componentes.

A estrutura do objeto composto é também usada para estabelecer dependências gráficas entre atributos dos diferentes objetos. Por exemplo, a posição de uma barra de menus na janela depende da região ocupada pela janela. Quando a janela é movida ou reformatada (por exemplo, tamanho alterado), esta alteração também acontece com a barra de menus associada a esta janela.

Um simples exame dos objetos de interação presentes nas aplicações atuais permite a constatação de que quase todos são objetos compostos, uma vez que dificilmente existe um único tipo de objeto visualmente distinto na tela mas sim uma combinação de vários deles. **A definição de painéis**, portanto, **é crucial para o projeto de boas IUs.**

Um painel é definido como uma instância de uma subclasse da classe painel. A característica principal desta classe é que, ao contrário das outras classes de objetos genéricos, seus atributos podem ser de dois tipos:

- **atributos componentes**, referentes a objetos de interação componentes e

- **atributos comuns**, referentes a objetos de tipos primitivos (inteiro,"string", etc) diferentes dos objetos de interação.

Um atributo do primeiro tipo consiste de uma referência a uma instância de alguma classe de objeto de interação definida, inclusive classes painéis. A definição de classes de objetos de IU é descrita no capítulo 7 e algumas classes presentes na FIU Canônica estão no Anexo C.

Um objeto genérico pode ser então:

- um objeto primitivo, que pode ser definido totalmente sem referenciar outros objetos genéricos ou

- um objeto composto através da combinação de outros objetos genéricos.

6.4 Modificação da arquitetura da FIU Canônica visando suporte a uma "toolkit"

Uma vez que a arquitetura espelha as estruturas de diálogos interno e externo, é natural que a decisão de alterar o tipo de controle, transformando a FIU Canônica de "toolbox" em "toolkit", acarrete alterações na arquitetura. Por isto, uma arquitetura, se quer possibilitar esta transformação, deve ser projetada de forma que tal

alteração só ocasione pequenas modificações no diálogo interno, ou seja, no protocolo de comunicação entre a aplicação e a IU.

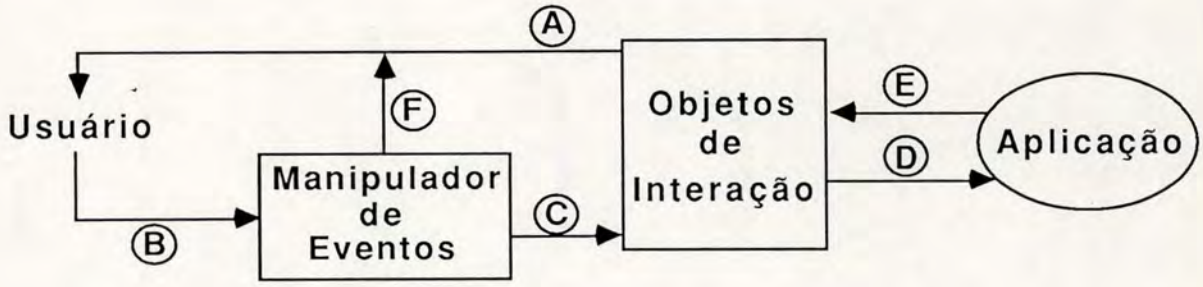

Figura 6.6 - Arquitetura para suporte
a uma "toolkit" Canônica

A arquitetura para suporte a uma **"toolkit"** canônica é apresentada na figura 6.6. As letras de A a F correspondem a comunicações entre os agentes e objetos de IU. Esta arquitetura apresenta um comportamento descrito pelos seguintes passos:

1) Os objetos de interação são apresentados ao usuário (comunicação A);

2) O usuário manipula os objetos através de ações (comunicação B); durante a ação do usuário é fornecido o "feedback" dinâmico (comunicação F) para eventos e movimentos do cursor do teclado e "mouse".

3) As ações são interpretadas pelo manipulador de eventos que gera eventos compreensíveis pelos objetos de interação (comunicação C);

4) Como o controle, usando-se uma "toolkit", é externo, se o evento solicitado pelo usuário é um comando, a aplicação é acionada através de chamadas às suas funções (comunicação D); se o evento não é solicitação de comando, o próprio objeto de interação trata-o adequadamente;

5) Os resultados/retornos a serem apresentados ("feedback bruto") são enviados para os objetos de interação (comunicação E);

6) Retorno ao passo 1, com a modificação da apresentação do objeto de interação ("feedback" decorado).

Note-se que , numa arquitetura para suporte a uma "toolkit" Canônica (figura 6.6), **as seguintes comunicações** na descrição do diálogo interno **foram modificadas** em relação à arquitetura da "toolbox" (figura 6.2):

Comunicação C (figura 6.6) = eventos são direcionados a objetos de interação, que podem tratá-los ou passá-los para a aplicação; **e não mais** são chamados diretamente pela aplicação (figura 6.2);

Comunicação D (figura 6.6) = eventos que a aplicação deve tratar, dados e chamadas a rotinas da aplicação; **e não mais** mensagens da aplicação aos objetos de interação (figura 6.2);

6.4.1 Modificação no Manipulador de Eventos

Numa "toolkit", o manipulador de eventos precisa tratar toda a entrada e ainda identificar para qual objeto de interação deve ser notificado o evento. Isto acarreta um tratamento adicional.

Uma visão detalhada de como um manipulador de eventos identifica os objetos receptores dos eventos (ações/comandos) do usuário pode ser vista na figura 6.7.

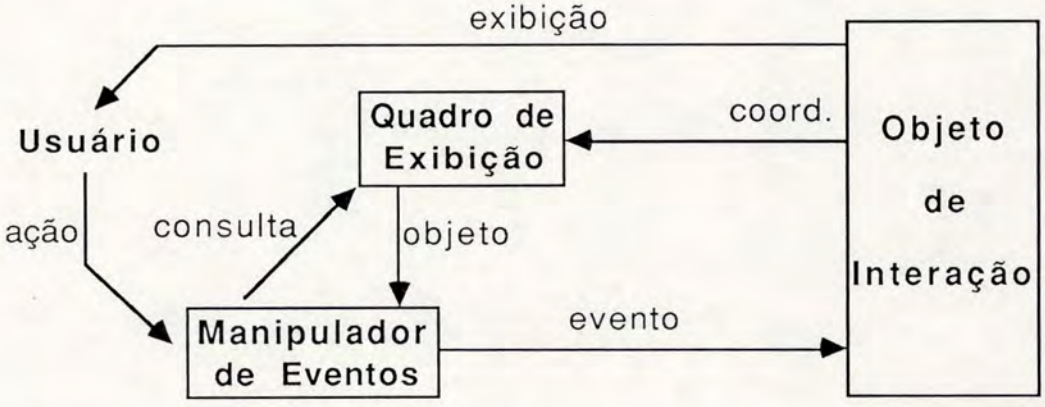

Figura 6.7 - Identificação de receptor de evento

A comunicação entre os manipuladores de eventos e objetos de interação é feita via um objeto chamado **quadro de exibição.** A proposta deste objeto é armazenar as coordenadas dos objetos de interação exibidos na tela. As coordenadas do objeto são registradas no quadro via

mensagem a cada (re)exibição. Estas coordenadas só podem ser atualizadas/alteradas por mensagens dos objetos de interação mas podem ser consultadas pelos manipuladores de eventos, para identificar a qual objeto um evento está associado.

Cada vez que um objeto é exibido, é enviada uma mensagem (**coord** na figura) ao quadro informando as coordenadas do objeto. O quadro atualiza sua descrição dos objetos visíveis na tela. O manipulador de eventos, quando recebe uma entrada do usuário, consulta o quadro (**consulta** na figura) e identifica para qual objeto (**objeto** na figura) será direcionado o evento.

Teoricamente, é como se houvesse uma associação pixel-objeto, ou seja, uma associação de qual pixel pertence a qual objeto. Em verdade, os métodos do quadro realizam os cálculos de interpolação necessários para descobrir a qual objeto está associada a entrada.

A noção de quadro é também proposta em /SIB 86/ com o nome de "container", mas com outra finalidade: propiciar a troca de informações entre a IU e a aplicação. Um mecanismo similar para permitir o tratamento de entrada em alto nível é o "link", descrito em /COU 85/.

6.5 Considerações

É possível e de fato desejável construir uma aplicação usando **somente** objetos de IU para comunicação com o usuário - sem nenhum outro tipo de E/S controlada ou

implementada pelo programa. O uso de objetos de IU oferece aos usuários projetistas uma série de benefícios:

1- Os objetos de interação possibilitam um protocolo de diálogo interno consistente aos usuários projetistas. Esta consistência reduz o tempo de aprendizado do modelo de IU utilizado e permite que o código relativo à IU seja mais fácil de entender e usar;

2- Objetos de interação formam uma estrutura comum e uniforme para diversos conjuntos de aplicações e plataformas (ambientes, hardware e sistemas operacionais), além de servir como um conjunto básico de padrões bem documentados para IUs em sistemas de aplicação compostos de muitos programas separados;

3- O conjunto dos objetos de interação cobre as mais comuns necessidades de uma IU, mesmo para IUs de manipulação direta, permitindo ao implementador dirigir sua atenção às funções do programa de aplicação;

4- Os objetos de interação são blocos para a construção de IU que já estão devidamente testados e depurados, aliviando os implementadores da tarefa de verificar se a IU comporta-se adequadamente e propiciando-lhes mais tempo para testar a aplicação. Isto é particularmente importante se considerarmos a complexidade de alguns objetos e quão difícil seria codificá-los do início.

7 O MODELO REPRESENTACIONAL CANONICUS

Neste capítulo o modelo de representação de IUs (MRIU) da FIU Canônica, o Canonicus, é apresentado. A adaptação da FIU Canônica a diferentes FIUs através do Canonicus é também discutida.

7.1 Introdução

Para a definição do MRIU para a FIU Canônica, foram avaliados três modelos representacionais (MRIUs) clássicos: o MRTE, o MGLC e o Modelos de Eventos.

MRTE é útil quando a metáfora é conversacional, com muito reconhecimento sintático ou com muitos modos (cada estado da rede corresponde na verdade a um modo). Os diagramas MRTE para IU muito grandes, complexas ou concorrentes tendem a ficar muito confusos e difíceis de criar/modificar, requerendo muitos arcos para cada estado. Além disso, todas conexões entre IU e a aplicação são feitas através de uma abundância de variáveis globais e todos os estados necessitam de arcos explícitos para manipulação de exceções (entradas erradas do usuário, por exemplo) e uso de comandos universais como "Help", "Cancel" ou "Undo".

O modelo **MGLC** é bom para definições de IUs conversacionais ou modais, mas são deficientes em IUs não-modais e gráficas por razões similares às apresentadas para o MRTE. Além disto, não fornece maneiras adequadas de

prover "feedback" semântico ou valores "default" porque não há modo das rotinas da aplicação afetarem o reconhecedor.

O **Modelo de Eventos** é o que possui maior poder descritivo, ou seja, possibilita que uma maior diversidade de IUs possam ser descritas por ele /GRE 86/. Além disso, Green mostrou que algoritmos eficientes para converter MGLC e MRTE para o Modelo de Eventos podem ser construídos /GRE 86/. Por isto, o Modelo de Eventos foi escolhido para ser a base da representação usada na FIU Canônica.

7.2 Um Modelo de Eventos Orientado a Objetos: O Canonicus

A motivação para o uso do PDOO na modelagem de eventos surgiu primeiramente com o Smalltalk /GOL 83b/. Além disto, Anson usou LOO para descrição de diálogos baseados na idéia de eventos /ANS 82/ e Cardelli produziu a linguagem Squeak para processar a entrada via "mouse" e teclado /CAR 85/. Squeak é baseada em processos e mensagens entre processos. Os processos são similares a manipuladores de eventos e mensagens servem a mesma proposta que eventos.

O MRIU proposto para a FIU Canônica, o modelo CANÔNico para Interface Com o USuário (abreviado **Canonicus**), é a modelagem no PDOO do Modelo de Eventos.

O modelo Canonicus tem as seguintes peculiaridades:

a) os manipuladores dos eventos (do Modelo de Eventos) são os **objetos de IU;**

b) os eventos (do Modelo de Eventos) correspondem a **mensagens** entre os agentes (usuário ou aplicação) e os objetos de IUs;

c) as mensagens ativam direta ou indiretamente os métodos definidos dos objetos e seu comportamento é pré-estabelecido;

d) o encapsulamento de atributos e operações evita que uma alteração num objeto seja propagada indevidamente: toda propagação é via mensagens;

e) as mensagens permitidas são apenas as previstas no protocolo dos objetos de IU.

Com esta abordagem de modelagem é possível, através de mensagens (de solicitação e retorno), especificar **qualquer** sintaxe necessária ao diálogo interno com uma aplicação. Isto permite a utilização do PDOO tanto para aplicações desenvolvidas segundo o PDOO quanto para aplicações desenvolvidas segundo o paradigma convencional /DAN 87/.

7.3 Explicação do Canonicus

O MRIU Canonicus possui uma notação para a descrição dos aspectos estáticos (composição e apresentação, por exemplo) e dos aspectos dinâmicos (comportamento através de operações) da IU. Esta notação é baseada na linguagem orientada a objetos **Eiffel**, cujas características principais foram sumarizadas no capítulo 5. A sintaxe da notação do Canonicus (doravante referenciada simplesmente como Canonicus) é então um subconjunto da

sintaxe de Eiffel com pequenas modificações. A gramática da notação está descrita no Anexo F.

Descrever uma IU no Modelo Canonicus é descrever as classes (genéricas e concretas) de objetos de IUs. Conseqüentemente, para operar sobre a IU definida no Modelo Canonicus deve-se instanciar objetos das classes definidas e manipulá-los de acordo com seu protocolo de mensagens.

7.3.1 Definição de Classes

A definição de uma classe consiste de :

- o nome da classe (**Class**);
- o nome da sua superclasse, se houver (**Inherit**);
- seu protocolo (**Export**) e
- suas características (**Feature**).

As **características** da classe descrevem os atributos e seus tipos e as rotinas e parâmetros associados. O **protocolo** (o nome usual é interface mas foi trocado para evitar confusões) lista os nomes das características acessíveis fora da classe, ou seja, a comunicação da classe com o exterior. Características não presentes no protocolo são denominadas **encapsuladas.**

Um exemplo de definição de classe é mostrado a seguir.

```
Class Exemplo
Inherit Objeto
Export
     atr2, rot1
Feature
atr1 : INTEGER is 50;
atr2 : Outra_Classe;
rot1 ( par1, par2 ) : INTEGER;
rot2 ( par3, par4 );
```

Uma classe de nome Exemplo foi definida. Sua superclasse é a classe Objeto. O seu protocolo lista as características atr2 e rot1 como acessíveis ao exterior. Deve-se notar que rotinas e atributos não são distingüidos no protocolo, segundo o princípio da referência uniforme /MEY 88/.

Em **Feature**, a descrição de um atributo consiste na declaração de um nome associado a um tipo. Um tipo pode ser primitivo (alguns dos pré-definidos INTEGER, REAL, CHAR, BOOLEAN) ou uma referência a uma classe (que denota referência a um objeto, instância da classe referenciada). Os atributos atr1 e atr2 são declarados respectivamente com tipos INTEGER e Outra_Classe. Usando-se Canonicus para descrever as classes concretas de objetos de IU, os tipos pré-definidos podem ser também os tipos utilizados para as FIUs específicas.

Os atributos podem ser inicializados na definição. Estes valores podem ser implícitos (inicializações pré-definidas para os tipos) ou explícitos. Os tipos das FIUs específicas devem ser inicializados explicitamente.

As inicializações pré-definidas para os tipos seguem as inicializações automáticas asseguradas em Eiffel /MEY 88/:

TIPO	VALOR INICIAL
INTEGER	0
BOOLEAN	FALSE
STRING	cadeia de caracteres nula
CHAR	caractere nulo
REAL	0
Classes	NIL (referência nula)

As inicializações explícitas possuem a seguinte sintaxe:

<atributo> : <tipo> **is** <constante>

Na classe exemplo, o atr1 é explicitamente inicializado com valor 50 e o atr2 implicitamente com NIL.

As operações descritas podem corresponder a funções, que retornam um valor, ou em caso contrário procedimentos ("procedures"). Portanto, rot1 é uma função que retorna INTEGER e rot2 é um procedimento. No exemplo, tanto rot1 quanto rot2 são rotinas deferidas, isto é, definidas sem a descrição da sua implementação.

7.3.2 Definição das Classes Genéricas

As classes genéricas descrevem o nível abstrato dos objetos de IU.

A definição das classes genéricas da FIU Canônica usando o Canonicus tem as seguintes peculiaridades:

a) O termo **"Generic"** antecede o léxico **"Class"**, indicando que o que segue é a definição de uma classe genérica;

b) Todos as rotinas são **deferidas**, isto é, não apresentam descrição de implementação.

A tarefa do projetista de interface no nível abstrato é escolher os objetos de IU que serão usados, suas características e seus protocolos e particularizá-los, isto é, torná-los específicos para as tarefas da aplicação. Para isto, o projetista de interface pode usar as classes genéricas pré-definidas na FIU Canônica ou definir novas classes genéricas compondo-as (como painéis) a partir das pré-definidas.

Estas duas maneiras são denominadas respectivamente particularização dinâmica e particularização estática.

A **particularização dinâmica** consiste em usar, para a definição da IU, apenas as classes pré-definidas e deixar sua particularização a cargo da aplicação, ou seja, do programador. Para isto o programador da aplicação deve usar o protocolo dos objetos para atribuir valores aos atributos e definir os objetos específicos à aplicação desejados, seguindo alguma política definida pelo projetista de interface. A particularização dinâmica, apesar da dificuldade de uso pelo programador de aplicação, apresenta a vantagem da flexibilidade, permitindo uma

mudança dos objetos (uma nova particularização) no decorrer do programa.

A **particularização estática** consiste em criar, a partir das classes pré-definidas (classes primitivas), novas classes mais particulares, voltadas para as tarefas da aplicação. Isto é feito através do mecanismo de herança ou da composição de painéis. Cada uma desta classes particulares tem a maior parte de seus atributos (tipicamente os que não referenciam outras classes) inicializados explicitamente. Os atributos que referenciam outras classes de IU precisam ser inicializados no decorrer da execução.

Desta forma, um objeto de IU particularizado é uma instância (possivelmente única) de uma classe. A particularização estática possibilita um uso facilitado dos objetos de IU pelo programador da aplicação, pois a particularização do objeto já está quase pronta a tempo de definição.

Em suma, a definição de IU pelo projetista de interface no nível abstrato inclui:

a) a atribuição de valores default a atributos das classes genéricas, a tempo de definição das classes (particularização estática) e/ou

b) o estabelecimento de uma política de uso do protocolo para a particularização pelo programador de aplicação, a tempo de uso pela aplicação (particularização dinâmica).

Exemplos de definição de classes genéricas da FIU Canônica usando o Canonicus estão presentes no Anexo C.

7.3.3 Definição das Classes Concretas

As classes concretas descrevem o nível concreto dos objetos de IU.

A definição das classes concretas usando o Canonicus tem por proposta:

a) encapsular no nível concreto as características de apresentação dos objetos de IU;

b) servir para a descrição da implementação dos métodos deferidos das classes genéricas;

c) servir de base para o processo de tradução da FIU Canônica para uma FIU específica, descrevendo o mapeamento entre as duas.

A definição das classes concretas da FIU Canônica usando o Canonicus tem as seguintes peculiaridades:

a) O termo **"Concret"** antecede o léxico **"Class"**, indicando que o que segue é a definição de uma classe concreta;

b) Os atributos podem possuir tipos relacionados à FIU específica;

c) Todos os métodos possuem implementação associada.

A tarefa do projetista de interface no nível concreto é definir a apresentação dos objetos de IU usados no nível abstrato e os detalhes de sua concretização em uma FIU específica. Isto corresponde à definição de uma classe concreta.

Como uma classe concreta é subclasse de uma classe genérica, todos os atributos da superclasse genérica são herdados pela concreta, podendo também ser redefinidos ("**overriden**"). Os atributos adicionais presentes na classe concreta são tipicamente atributos necessários para a apresentação dos objetos e/ou à execução dos métodos.

A definição da implementação dos métodos deferidos das classes genéricas é feita através do mapeamento de um método para um conjunto (possivelmente unitário) de rotinas de uma FIU, o qual corresponde à implementação das tarefas do método.

A notação Eiffel para descrever a implementação de rotinas tem a seguinte sintaxe :

```
<nome_rotina> ( <lista_de_parâmetros> ) is
local <lista_de_decl_de_var_locais>
do
     <implementação>
end
```

e é interpretada da seguinte maneira:

<nome_rotina> é o nome do método, herdado da classe genérica, onde era deferido;

<lista_de_parâmetros> é opcional e envolve atributos presentes na classe genérica e também herdados pela concreta;

<lista_de_decl_de_var_locais> é um conjunto de declarações das variáveis locais usadas na <implementação>;

<implementação> é um conjunto de operações que visa executar as tarefas especificadas para a rotina <nome_rotina>.

No Canonicus, este conjunto pode conter rotinas da FIU específica (doravante denominadas **rotinas FIU**) e rotinas auxiliares necessárias para a obtenção do mapeamento total quando as rotinas FIU não são suficientes. As rotinas **Conversao_X** e **Conversao_Y** no exemplo EXR1 são exemplos de rotinas auxiliares. Uma rotina FIU só possibilita mapeamento total quando possui uma correspondência 1:1 com a rotina <nome_rotina>.

Exemplo EXR1:

```
Mover(x,y) is
local x1:INTEGER;
      y1:INTEGER;
do
     x1 := Conversao_X (x);
     y1 := Conversao_Y(y) ;
     MoveWindow(x1,y1);
end
```

No exemplo EXR1, a rotina MoveWindow é uma rotina FIU que não implementa todas as funções previstas para Mover da FIU Canônica, não possibilitando a correspondência 1:1. Para prover esta correspondência, as rotinas auxiliares **Conversao_X** e **Conversao_Y** são usadas, no caso, para tarefas de conversão de coordenadas não feitas pela MoveWindows.

Conversao_X e **Conversao_Y**, então, não são operações da FIU Canônica mas rotinas auxiliares que complementam as tarefas das rotinas FIUs nas classes concretas para prover comportamento da classe genérica.

Além disto, operações auxiliares de atribuição podem ser feitas para executar as mudanças nos valores dos atributos (herdados ou não) dos objetos manipulados, conforme exemplo EXR2 abaixo.

Exemplo EXR2:

```
Selecionar (j) is
do
     SelectWindow(j);
     Selecionada := TRUE;
end
```

Uma rotina auxiliar pode ser uma rotina desenvolvida pelos usuários projetistas. Neste caso ela é definida segundo a cláusula external do Canonicus:

```
external <nome_rotina> ( <lista_de_parametros> )
         name <nome_local>
         language <nome_linguagem>
```

Uma rotina auxiliar é definida da mesma forma que uma rotina FIU (nome e parâmetros) acrescida de informações sobre a linguagem em que foi desenvolvida (**language**) e sobre o local onde encontra-se o código da rotina (**name**). O parâmetro **<nome_linguagem>** deve ser o nome de uma das possíveis linguagens previstas para uso com as FIUs. Tipicamente, a Mac Toolbox está vinculada a Pascal e o MS-Windows a Pascal ou C. O parâmetro **<nome_local>** pode designar tanto o nome do arquivo (se código fonte) ou o nome da biblioteca (se código objeto) onde a rotina se localiza.

Eventualmente, podem haver diferentes rotinas de uma mesma FIU que implementam a mesma função de um método deferido. A decisão de usar uma ou outra cabe ao projetista de interface. No entanto, a descrição do mapeamento do método deferido pode ser feita para as diferentes alternativas de implementação com a seguinte notação:

```
Remover () is
do
     CloseDialog();
     |
     DisposeDialog();
end
```

onde o símbolo "|" separa as alternativas. A ordem de definição implica prioridade para efeito de tradução, ou seja, o tradutor usa prioritariamente a primeira alternativa. O projetista de interface decide qual usar decidindo a ordem de definição.

Exemplos de definições de classes concretas para a Mac Toolbox e para o MS-Windows usando o Canonicus estão respectivamente apresentados nos Anexos D e E.

7.3.3.1 Classes concretas para novas FIUs

As classes concretas definidas nos apêndices D e E foram definidas para 2 FIUs específicas. No entanto, a FIU Canônica é genérica e deve permitir a definição de um conjunto de classes concretas para cada nova FIU subjacente que se deseja usar.

A dificuldade de definir as classes concretas está diretamente relacionada à dificuldade de:

a) definir atributos da FIU específica na classe concreta; e

b) estabelecer a correspondência 1:1 entre rotinas de classes genéricas e rotinas de classes concretas da FIU Canônica.

Como, no Canonicus, os atributos da classe concreta podem ser de tipos relacionados à FIU específica, a primeira dificuldade é praticamente inexistente.

A segunda dificuldade é resolvida pela habilidade do projetista de interfaces em identificar e estabelecer a correspondência desejada..

Por definição do Canonicus, uma rotina da classe genérica pode corresponder a um conjunto (possivelmente unitário) de rotinas FIU nas classe concreta. Uma correspondência 1:1 significa que este conjunto é unitário.

No caso da correspondência ser realmente 1:1, a tarefa do projetista de interfaces é trivial: as rotinas FIU suprem toda a funcionalidade da rotina da classe genérica correspondente. Como as rotinas definidas para as classes genéricas são as operações mais freqüentes e importantes utilizadas sobre os objetos de IU /LIN 89/, este é o caso típico.

No caso da correspondência não ser 1:1, o projetista de interfaces deve desenvolver rotinas auxiliares para prover esta correspondência. O Canonicus permite a definição de rotinas auxiliares e combinação de rotinas auxiliares e rotinas FIU em qualquer grau de complexidade necessário para a tarefa do projetista. No entanto, fica a critério do projetista analisar se vale a pena desenvolver rotinas e combinações muito complexas quando esta correspondência não for tão direta.

7.3.4 Uso das Classes do Canonicus

As classes genéricas definidas pelo projetista de interface são usadas pelo programador de aplicação para estabelecer o comportamento da IU. O uso das classes segue tipicamente dois passos:

- a criação de instâncias das classes e a inicialização de seus atributos e
- a manipulação das instâncias (operações sobre os objetos das classes) para fornecer o comportamento da IU.

Antes de serem usadas, no entanto, as classes de IU para uma aplicação precisam ser declaradas.

7.3.4.1 Declaração na aplicação

As classes dos objetos de IU usados pela aplicação devem ser declaradas. A declaração serve de documentação para os usuários projetistas e de orientação para os procedimentos de tradução.

A declaração das classes usadas por uma aplicação é feita através da cláusula **USE <lista_de_classes>**, inserida no início do fonte da aplicação. O parâmetro **<lista_de_classes>** consiste da lista das classes genéricas de objetos de IU usadas.

7.3.4.2 Criação de instâncias e Inicialização

Instanciar objetos de IU é fundamental para a sua manipulação. Tem-se, a tempo de compilação, um conjunto de classes de IU e, a tempo de execução, um conjunto de objetos de IU - instâncias destas classes - que podem acessar atributos e receber mensagens.

Objetos de IU apenas são criados como resultado da execução da rotina Criar sobre o nome da classe a que pertencem. Por exemplo,

```
j1:=Janela.Criar()
```

é a criação de um objeto de nome j1, instância da (na verdade instância de uma subclasse concreta da) classe Janela.

A operação Criar tem os seguintes efeitos:

1) Inicialização de todos os atributos dos objetos com os valores de inicialização (pré-definidos ou explícitos) correspondentes;

2) Execução dos procedimentos para a rotina Criar, conforme definida na classe, com os parâmetros reais dados.

É importante observar que a definição dos procedimentos nas classes concretas (efeito 2) existe para todas operações mas a inicialização dos atributos (efeito 1) é uma ação exclusiva da operação Criar, que deve ser provida explicitamente pelo tradutor da FIU Canônica, diferenciando esta operação das demais.

A inicialização dos objetos particularizados é feita de acordo com o modo de particularização.

Com a **particularização estática**, o programador de aplicação apenas cria os objetos (com a mensagem Criar). Os objetos já possuem valores significativos previamente atribuídos aos atributos e estão prontos para serem manipulados. As referências a outros objetos (referências a outras classes na definição) devem ser inicializadas quando os objetos referenciados forem criados.

Com a **particularização dinâmica**, os valores dos atributos dos objetos devem ser atribuídos, no programa, através do uso das operações de atribuição e definição disponíveis no protocolo dos objetos.

7.3.4.3 Operações de manipulação sobre as instâncias

Tipicamente, o programador manipula os objetos de IU através da ativação de seus métodos. Para isto, utiliza o protocolo definido nas classes dos objetos.

Em linguagens orientadas a objeto, qualquer operação é relativa a um certo objeto. Para invocar uma operação, o objeto a que ela se aplica é especificado qualificando-se o nome da operação, isto é, escrevendo-se seu nome no lado esquerdo do nome da operação. O caractere "." é o separador dos nomes. A notação:

Objeto.operação (par1, par2)

significa "aplique **operação**, com os parâmetros **par1** e **par2**, a **objeto**". Os parâmetros podem estar ausentes

se a rotina foi definida assim. Em programação convencional, a notação correspondente seria : **operação (objeto, par1, par2)**, isto é, o objeto seria um dos argumentos da operação.

Como no Canonicus todas as operações (exceto Criar, explicada anteriormente) são relativas aos objetos, para exibir ou remover a janela j1 basta respectivamente invocar

```
j1.exibir();
```

ou

```
j1.remover();
```

no decorrer do programa.

Para uma operação ser significativa, o objeto deve existir, ou seja, sua referência não pode ser nula. Isto é verdade não só para a execução da rotina mas também para acesso aos atributos.

Dentro de uma classe muitas vezes, o objeto corrente ao qual operações se aplicam usualmente permanece implícito: referências não qualificadas de atributos nas operações significam "atributos do objeto corrente". A variável especial **Current** pode ser usada, se necessário, para denotar explicitamente este objeto. Então um atributo **selecionada** não qualificado usado no corpo de uma operação é equivalente a **Current.selecionada** (ver exemplo EXR2).

8 EXEMPLO DE USO DA FIU CANÔNICA

8.1 Descrição do programa interativo Exemplo

Este é um programa exemplo escrito em Pascal apresentado em /APP 85/. Embora rudimentar, este exemplo é prático o suficiente para mostrar as diferenças e semelhanças no uso das FIUs. O nome do programa é **Exemplo** e por este nome será referenciado nesta seção.

Uma porção de comentários visam esclarecer a função do programa.

O programa **Exemplo** exibe uma janela onde o usuário pode entrar e editar texto. Na janela há três menus: um menu **Desk** para escolha de "desk acessories"; um menu **File**, contendo somente um comando **Quit**; e um menu **Edit**, contendo os comandos básicos de edição: **Undo**, **Cut**, **Copy**, **Paste** e **Clear**.

O usuário pode mover a janela pela tela, deslocando-a para a posição que achar adequada. No entanto, não pode fechá-la, paginá-la ("scroll"), e reconfigurar seu tamanho ("resize"). Porque o menu **File** contém somente um comando **Quit**, o documento não pode ser salvo ou impresso. O comando **Quit** encerra o programa. No menu **Edit** o comando **Undo** está desabilitado para a aplicação mas não para algum "desk acessory" chamado.

8.2 O programa interativo Exemplo usando o Mac Toolbox

O programa **Exemplo** implementado usando o Mac Toolbox usa, além do código Pascal mostrado a seguir, um arquivo de recursos (também apresentado em /APP 85/) que inclui os dados listados na figura 8.1. O programa usa os números na segunda coluna para identificar os recursos.

Recurso	Número do Recurso	Descrição
Menu	128	Menu com título "DESK" e sem comandos
Menu	129	Menu com título "File" e com o comando "Quit"
Menu	130	Menu com título "Edit" e com os comandos "Cut", "Copy", "Paste" e "Clear", nesta ordem e com uma linha divisória entre o "Undo" e o "Cut"
Window Template	128	Janela tipo Documento sem "size box"; Coordenada do ponto superior esquerdo (50,40) e do ponto inferior direito (300,450); título "Exemplo"; e sem "close box"

Figura 8.1 - Recursos da Mac Toolbox para o programa Exemplo

Cada recurso menu também contém um identificador de menu ("menu ID") que é usado para identificar o menu do qual foi escolhido um comando pelo usuário. Para os três

menus de **Exemplo**, este identificador é igual ao número do recurso.

Para criar um arquivo de recursos com os conteúdos acima, pode-se usar o Editor de Recursos disponível no ambiente Mac.

O programa começa com uma cláusula **USES** que especifica todos os arquivos necessários para uso do Pascal. A seguir, declarações de constantes e variáveis com os tipos de dados usuais do Pascal ou tipos definidos nos arquivos da Mac Toolbox (como Rect e WindowPtr). Variáveis usadas no programa mas não declaradas são globais definidas no QuickDraw.

Após declaradas as rotinas **SetUpMenus** e **DoCommand**, o programa começa com uma seqüência de inicialização padrão. Toda aplicação Mac necessita fazer estas mesmas inicializações na ordem mostrada. Seguem-nas as inicializações específicas do programa. Isto inclui a inicialização dos menus e da barra de menus (chamando **SetUpMenus**) e criando a janela documento da aplicação (lendo sua descrição do arquivo de recursos e exibindo-a na tela).

O núcleo de toda aplicação Mac é o "loop" principal de eventos, que repetidamente chama o Gerente de Eventos para obter os eventos e tratá-los apropriadamente. O evento mais comum é o "click" do botão do "mouse"; dependendo de onde ele ocorreu, como reportado pelo Gerente de Janelas, o programa Exemplo pode executar um comando, mover a janela documento, tornar a janela ativa ou passar o

evento para um "desk acessory" (abreviado **DA**) . A rotina **DoCommand** encarrega-se de executar um comando da aplicação, a partir da informação recebida pelo Gerente de Menus para determinar qual comando executar.

Além dos eventos resultantes diretamente de ações do usuário, há os eventos decorrentes destas ações. Por exemplo, quando uma janela muda de ativa para inativa ou vice-versa, o Gerente de Janelas comunica ao Gerente de Eventos para reportar isto à aplicação. Um processo similar acontece quando uma janela precisa ser parcial ou totalmente redesenhada ou atualizada. O programa simplesmente responde a cada evento quando notificado.

O "loop" de eventos termina quando o usuário encerra o programa - neste caso, através do comando **Quit.**

Os conceitos (e as siglas) da Mac Toolbox estão descritos no capítulo 3. As linhas do programa estão numeradas e serão referenciadas nos comentários.

```
**********************************************************
1    PROGRAM Exemplo;

{ Exemplo :  uma aplicação exemplo escrita sobre a Mac
Toolbox. Uma janela de tamanho fixo é exibida,  onde o
usuário pode editar e manipular texto. }

5     USES ($U Obj/MemTypes ) MemTypes,
                    {tipos para gerência de memória}
           ($U Obj/QuickDraw) QuickDraw,
                    {interface com QuickDraw}
           ($U Obj/OSIntf    ) OSIntf,
                    {interface com sistema operacional}
           ($U Obj/ToolIntf ) ToolIntf;
                    {interface com Toolbox}
```

```
{*************************************************}

10    CONST DeskId   = 128; {números de recursos}
            FileId   = 129;
            EditId   = 130;
            windowId = 128;

            DeskM = 1;  {índices de menus no array myMenus}
15          FileM = 2;
            EditM = 3;

            menuCount    = 3; {número total de menus}

            undoCommand  = 1; { numeros de itens de menus }
            cutCommand   = 3; { identificando comandos no }
20          copyCommand  = 4; { no menu Edit              }
            pasteCommand = 5;
            clearCommand = 6;
{*************************************************}
      VAR   myMenus: ARRAY [1..menuCount] OF MenuHandle;
               {array de handles para menus }
            dragRect: Rect;
               {ret. limite da janela de desenho}
25          txRect: Rect;
               {ret. limite para texto na janela}
            textH: TEHandle;   {handle para texto}
            theChar: CHAR;     {caractere digitado}
            extended: BOOLEAN;
               {TRUE se junto com tecla SHIFT}
            doneflag: BOOLEAN;
               {TRUE se comando Quit}
30          myEvent: EventRecord;
               {informacao sobre evento}
            wRecord: WindowRecord;
               {informacao sobre a janela da aplicacao}
            myWindow: WindowPtr;
               {ponteiro para a janela}
            whichWindow: WindowPtr;
               {ponteiro para a janela onde mouse }
               {foi pressionado                   }
{*************************************************}
35    PROCEDURE SetUpMenus;
               { Inicializa os menus e a barra de menus }
      VAR i: INTEGER;
      BEGIN
      myMenus[DeskM] := GetMenu(DeskId);
               {lê menu Desk do arquivo}
40    AddResMenu(myMenus [DeskM], 'DRVR');
               {adiciona DAs ao menu}
      myMenus[FileM] := GetMenu(FileId);
               {lê menu File do arquivo}
      myMenus[EditM] := GetMenu(EditId);
               {lê menu Edit do arquivo}
```

```
      FOR i:=1 to menuCount DO
          InsertMenu(myMenus[i],0);
                    {instala menus na barra de menus}
45    DrawMenuBar;  {e a desenha                      }
      END; {SetUpMenus}

{*************************************************}

      PROCEDURE DoCommand (mResult: LONGINT);
                {Executa comando especificado por   }
                {mResult,  selecionado num menu     }
50    VAR theItem: INTEGER;
                {numero de item contido em mResult}
          theMenu: INTEGER;
                {numero de menu contido em mResult}
          name: Str255;
                {nome do desk acessory}
          temp: INTEGER;
                {variável temporária}
      BEGIN
55     theItem := LoWord(mResult);
       theMenu := HiWord(mResult);
       CASE theMenu OF

       DeskId:
         BEGIN
60       GetItem (myMenus[DeskM],theItem,name);
                {obtem nome do DA e chama}
         temp := OpenDeskAcc(name);
                {Desk Manager para chamar o }
                {DA (em outra janela)       }
         SetPort(myWindow);
                { restaura a janela }
         END;  {DeskId}

       FileId:
65       doneflag := TRUE; {encerrar programa }

       EditId:
          BEGIN
          IF NOT SystemEdit(theItem - 1)
          THEN
70          CASE theItem OF
                      cutCommand:   TECut(textH);
                      copyCommand:  TECopy(textH);
                      pasteCommand: TEPaste(textH);
                      clearCommand: TEDelete(textH);
75          END;   {Case theItem}
          END;  {EditId}
       END; {do Case theMenu }
       HiliteMenu (0);
       END; {DoCommand }
```

```
80 {*****************************************************}
      BEGIN  {Programa Principal }
{Inicialização Geral}
      InitGraf(@thePort);
               {Inicializa QuickDraw}
      InitFonts;
               {Inicializa FM }
85    FlushEvents(everyEvent,0);
               {Inicializa Ev. M.}
      InitWindows;
               {Inicializa WM}
      InitMenus;
               {Inicializa MM}
      TEInit;
               {Inicializa TextEdit}
      InitDialogs(NIL);
               {Inicializa DM}
90    InitCursor;
               {Inicializa cursor como uma seta}

{Inicialização Específica}
      SetUpMenus;
      WITH screenBits.bounds DO
      SetRect(dragRect,4,24,right-4,bottom-4);
95    doneflag := FALSE;
      myWindow:=GetNewWindow(windowID,@wRecord,POINTER(1));
      SetPort(myWindow);
      txRect := thePort^.portRect;
      InsetRect(txRect,4,0);
100   textH := TENew(txRect,txRect);

{Loop de Eventos Principal}
      REPEAT
        SystemTask;
        TEIdle(textH);
105     IF GetNextEvent(everyEvent, myEvent)
        THEN
          CASE myEvent.what OF

          mouseDown:
            CASE FindWindow(myEvent.where,whichWindow) OF

110           inSysWindow:
                SystemClick(myEvent,whichWindow);

              inMenuBar:
                DoCommand(MenuSelect(myEvent.where));

              inDrag:
115           DragWindow(whichWindow,myEvent.where,dragRect);
```

```
            inContent:
              BEGIN
              IF whichWindow <> FrontWindow
              THEN SelectWindow(whichWindow)
120           ELSE BEGIN
                   GlobaltoLocal(myEvent.where);
                   extended :=   BitAnd(myEvent.modifier,
                                        shiftKey) <> 0;
                   TEClick(myEvent.where,extended,textH);
                   END;
125           END; (inContent}
          END; {mouseDown}

        keyDown, autoKey:

          BEGIN
          theChar :=   CHR(BitAnd(myEvent.message,
                           charCodeMask));
130       IF BitAnd(myEvent.modifiers,cmdKey) <> 0
          THEN DoCommand(MenuKey(theChar))
          ELSE TEKey(theChar,textH);
          END; {KeyDown, autoKey }

        activateEvt:

135       BEGIN
          IF BitAnd(myEvent.modifiers, activeFlag) <> 0
          THEN BEGIN
               TEActivate(textH);
               DisableItem(myMenus[editM],undoCommand);
140            END
          ELSE BEGIN
               TeDeactivate(textH);
               EnableItem(myMenus[editM],undoCommand);
               END;
145       END;{activateEvt}

        updateEvt:
          BEGIN
          BeginUpdate(WindowPtr(myEvent.message));
          EraseRect(thePort^.portRect);
150       TEUpdate(thePort^.portRect,textH);
          EndUpdate(WindowPtr(myEvent.message))[
          END;  {UpdateEvt}
        END;  {do case de eventos}

    UNTIL doneflag;
155 END.
{*****************************************************}
```

8.2.1 Comentários

A seguir, são tecidos alguns comentários visando esclarecer o comportamento global do programa e algumas de suas funções relacionadas especificamente com a IU.

Os números de linhas usados entre parênteses nos comentários referem-se às linhas do programa apresentado. Por exemplo (45) refere-se à linha 45 e (46-67) refere-se ao trecho compreendido entre as linhas 46 e 67.

Rotina SetUpMenus

A rotina **SetUpMenus** (35-46) lê as descrições de menus do arquivo de recursos para a memória e armazena seus "handles" (ponteiros indiretos para estruturas de dados), inicializando os menus e a barra de menus.

AddResMenu é uma rotina que lê nomes de recursos e os adiciona ao menu, usada freqüentemente para fontes e DAs. '**DRVR**' é um tipo de recurso para DAs (40).

Depois, menus (nomes na barra de menus) são associados a arquivos de recursos, que são lidos (41-42). A barra de menus é inicializada e desenhada (43-45).

Rotina DoCommand

O parâmetro **mResult** é o resultado da seleção de itens de menu (ver rotina **MenuSelect**). A chamada das funções associadas aos itens dos menus é feita após sua identificação. Esta identificação é feita através de dois comandos CASE: um para saber qual menu (57-77) e outro para saber o item selecionado no menu. No **Exemplo**, apenas o menu **Edit** tem mais itens (70-76).

O primeiro menu é o menu dos DAs (**Desk**), que são outras aplicações que rodam cada uma na sua janela. Para chamá-las, primeiro deve-se obter o nome do DA (60), passá-lo para o Desk Manager (61), que controla sua execução. Para retornar ao **Exemplo**, sua janela deve ser restaurada (62).

O segundo menu é o menu de arquivos (**File**), com **Quit** como a opção única (65).

O terceiro menu é o menu de edição (**Edit**). Os comandos de edição estão presentes tento para **Exemplo** como para os DAs. O teste (68) é para verificar se o item selecionado é um comando para o DA (e a aplicação ignora-o) ou para a aplicação, que deve tratá-lo adequadamente, chamando os procedimentos para manipulação de texto (70-75). Após a execução do procedimento selecionado, é desligado o realce ("highlight") do item selecionado (78).

Programa Principal

Após as inicializações, o corpo do programa é um "loop" de tratamento de eventos ("polling").

Nas inicializações globais (83-90) os gerentes da Mac Toolbox são inicializados. A ordem de inicialização é fixa: os anteriores são usados pelos posteriores.

A inicialização específica (92-100) diz respeito a uma série de providências a serem tomadas para associar a aplicação **Exemplo** a uma janela e preparar o ambiente de edição de texto:

- inicializar menus (92);
- estabelecer limites para movimentação da janela (93-94);
- criar janela (96);
- estabelecer porta gráfica da janela (97);
- preparar retângulo para edição de texto na janela (98-99);
- preparar recepção de texto (100).

O "loop" de eventos é iniciado com a obtenção do código do evento (105). A identificação do evento é feita num grande CASE sobre os tipos de eventos (107). A seguir, há a associação de um evento a seu tratamento pela aplicação (108-153). O fim do "loop" (154) ocorre quando a variável de controle de loop (29) tem seu valor TRUE ,

refletindo que o usuário escolheu a opção **Quit** do menu **File** (65).

Se o evento é um "click" do mouse (108), o tratamento depende da posição em o que o "click" foi dado . Isto também é identificado via CASE (109-126). Se o "click" foi dado fora da janela da aplicação (110), uma rotina para tratamento de DAs é chamada (111). Se o "click" foi dado na barra de menus (112), apresenta o menu correspondente e controla a seleção de itens, executando o selecionado (113). Se o "click" foi dado na barra de títulos da janela (114), conhecida por "drag region", movimenta a janela (115). Se o "click" foi dado na região de conteúdo da janela (116):

- verifica se a janela está ativa (118),
- ativando-a se não estiver (119),
- converte coordenadas do "mouse" para coordenadas da janela (121-122), e
- trata o evento (no **Exemplo**, seleção de intervalo de texto para edição).

Se o evento é causado por teclas pressionadas (127), deve-se testar (130) se estas teclas formam um código de função (ALT-P ou CTRL-Q, por exemplo) para seleção de menu (131) ou se são teclas comuns para entrada de texto (132).

Se o evento um evento de ativação (134), significa que uma janela foi tornada ativa ou inativa pelo usuário. Devem ser executados os procedimentos para

ativação de janela (138-140) ou desativação da janela (141-145).

Se o evento é um evento de atualização (146), significa que a visão da janela deve ser atualizada . O trecho de atualização é tipicamente limitado por comandos de início (148) e fim (151) de atualização.

8.3 O programa interativo Exemplo1 usando a FIU Canônica

O programa **Exemplo1** possui as mesmas funções do programa **Exemplo** e é também escrito em Pascal. A diferença é que **Exemplo1** é implementado usando a FIU Canônica e sua notação Canonicus, mais especificamente as classes genéricas (ver Anexo C) e concretas para o Mac Toolbox (ver Anexo D).

```
{*********************************************}
1    PROGRAM Exemplo1;

{  Exemplo1 :  uma aplicação exemplo escrita usando o
Canonicus. Uma janela de tamanho fixo é exibida,  onde o
usuário pode editar e manipular texto. }

5      USE Menu, Janela, Texto, Manip_Evento;

{*************************************************}

       CONST DeskId   = 128; {números de recursos}
             FileId   = 129;
             EditId   = 130;
10           windowId = 128;

{índices de menus no array myMenus}
             DeskM = 1;
             FileM = 2;
             EditM = 3;
```

```
            menuCount = 3;  {número total de menus}
{números de itens de menus identificando }
{comandos no menu Edit}
15          undoCommand  = 1;
            cutCommand   = 3;
            copyCommand  = 4;
            pasteCommand = 5;
            clearCommand = 6;

20 {*************************************************}
      VAR   myMenus: ARRAY [1..menuCount] OF INTEGER;
            doneflag: BOOLEAN;
               {TRUE se comando Quit foi escolhido}

 {*************************************************}
25    PROCEDURE SetUpMenus (Jan_Id : INTEGER);
   { Inicializa os menus e a barra de menus }

      VAR i: INTEGER;

      BEGIN
      myMenus[DeskM] := Menu_Popup.Criar (DeskId);
30              {lê menu Desk do recurso}
      myMenus[FileM] := Menu_Popup.Criar (FileId);
              {lê menu File do recurso}
      myMenus[EditM] := Menu_Popup.Criar (EditId);
              {lê menu Edit do recurso}
35    bar            := Barra_Menu.Criar ();
      bar.janela := Jan_Id;

      FOR i:=1 to menuCount DO
          bar.ins_item ( myMenus[i], i );
              {instala menus na barra de menus}
      mymenus [FileM].ins_item ('DRVR');
40    bar.exibir ();
      END; {SetUpMenus}

{*************************************************}
      PROCEDURE DoCommand (Evento, Jan: INTEGER);
{Executa comando especificado por evento de }
{selecao num menu}

45    VAR nome: Str255;
              {nome do desk acessory}
          abert : INTEGER;
              {valor da abertura de DA (nao usada)}
          idx: INTEGER;
              {indice do menu selecionado}

      BEGIN
      CASE Evento.Menu OF
```

```
50      DeskId:
           BEGIN
           nome := Obtem_item ( Evento. item);
           abert:= OpenDeskAcc (nome);
           myWindow.Exibir();
55         idx := DeskM;
           END; {DeskId}

        FileId:
           BEGIN
           doneflag := TRUE; {encerrar programa }
60         idx := FileM;
           END; {FileId}

        EditId:
           BEGIN
           IF myWindow.ativa = FALSE
65         THEN BEGIN
                nome := Obtem_item ( Evento. item);
                OpenDeskAcc (nome);
                END
           ELSE
70              CASE evento.Item OF

                   cutCommand:   buffer.cortar();
                   copyCommand:  buffer.copiar();
                   pasteCommand: buffer.colar();
                   clearCommand: buffer.remover();
75              END;   {Case evento.Item}
           idx:=EditM;
           END;  {EditId}
      END; {do Case theMenu }
      mymenus[idx].des_selecionar();

80    END; {DoCommand }
{*************************************************}

      BEGIN  {Programa Principal }


{Inicialização Específica}

      myWindow := Janela.Criar();
85    buffer := Texto.Criar();
      mevento : = Manip_Evento.Criar();
      SetUpMenus(myWindow);
      doneflag := FALSE;

{Loop de Eventos Principal}

90    REPEAT
        buffer.exibe_cursor();
        mevento.tipo := mevento.Qual_evento();
```

```
        CASE mevento.tipo OF

            SELECAO_MENU:
95              DoCommand(mevento, myWindow);

            MOVER_JANELA:
                mywindow.mover(mevento.posicaox,
                               mevento.posicaoy);

            SELECAO_JANELA:
                buffer.selecionar();

100         EDICAO_TEXTO:
                buffer.inserir(evento.char);

            SELECAO_TEXTO:
                buffer.selecionar(mevento.posicaox,
                                  mevento.posicaoy);

105         ATIVAR_JANELA:

                BEGIN
                buffer.ativar();
                mymenus[EditM].desabil_item(Undo);
                END;


110         DESATIVAR_JANELA:

                BEGIN
                buffer.desativar();
                mymenus[EditM].habil_item(Undo);
                END;


115         ATUALIZAR_JANELA:

                BEGIN
                mywindow.inic_atual();
                buffer.atualizar();
                mywindow.fim_atual();
120             END;  {atualizacao}

        END;  {do case de eventos}

      UNTIL doneflag;
      END. {do Exemplo1}

{*********************************************}
```

8.3.1 Comentários

O comportamento global do programa é o mesmo, tendo sido alteradas apenas algumas de suas funções relacionadas especificamente com a IU. O uso da FIU Canônica torna estas funções mais fáceis de usar, quase autoexplicativas, de forma que poucos comentários serão feitos.

Declarações

As classes genéricas usadas no programa devem ser declaradas (5). No caso da hierarquia de classes genéricas envolvendo **Menu**, **Barra_Menu** e **Menu_Popup**, declarou-se a mais abstrata (**Menu**, no exemplo).

As constantes e variáveis necessárias ao programa são declaradas nas linhas (7 - 23). Os nomes foram mantidos para facilitar a compreensão do programa. Algumas variáveis usadas antes para funções de IU são agora desnecessárias (como por exemplo, **whichWindows** e **txRect**). Isto evita confusão entre variáveis para a aplicação e variáveis para FIU.

Rotina SetUpMenus

A rotina **SetUpMenus** (25-41) continua com a mesma função de instalar e exibir uma barra de menus associada à janela. Agora. porém, um parâmetro identificando a janela foi acrescentado (25) , necessário para a inicialização de

atributos da barra. Os objetos menus (barra e cortinas associados) são criados (29-35).

Rotina DoCommand

Esta procedure tem agora dois parâmetros, que são identificadores de objetos: um manipulador de eventos e uma janela (43). Além disto, as variáveis locais da rotina diminuíram (45-47).

O manipulador de eventos contém em seus atributos a identificação do menu e do item selecionados. A janela é usada para o teste do terceiro menu.

No primeiro menu, a chamada dos DAs é feita obtendo-se o nome do DA (52), passando-o para o gerenciador de DAs (53), que controla sua execução. Para retornar ao **Exemplo1**, a janela de **Exemplo1** (**myWindows**) deve ser restaurada (54).

O segundo menu é o menu de arquivos, com **Quit** como a opção única (57-61), que encerrará o programa (59).

O terceiro menu é o menu de edição. Os comandos de edição estão presentes tanto para **Exemplo1** como para os DAs. O teste (64) é para verificar se a janela de **Exemplo1** está ativa ou não, implicando, respectivamente, no tratamento pelo programa **Exemplo1** do item selecionado, chamando os procedimentos para manipulação de texto (70-75), ou no tratamento por um DA (66-67).

Após a execução do procedimento associado ao menu, é desligado o realce ("highlight") do menu (um item da barra) selecionado (79). O item da barra é indicado pela variável **idx**, atribuída no corpo da rotina (55,60 e 76).

Programa Principal

No programa principal aparecem as grandes diferenças do programa **Exemplo1** usando a FIU Canônica em relação ao programa **Exemplo**, usando a Mac Toolbox.

A inicialização global que havia em **Exemplo** não é mais necessária em **Exemplo1**, uma vez que o tradutor da FIU Canônica para a Mac Toolbox deve gerar (baseado na cláusula **USE**) esta inicialização.

A inicialização específica consiste na criação dos objetos pertencentes às classes janela, texto e manipulador de eventos (84-86). A criação e inicialização dos menus é feita na rotina **SetUpMenus** (87). A variável **doneflag** é ainda o controle do término do programa (88). As operações que antes haviam em **Exemplo**, inicializando características de IU, são providas pelas classes concretas da FIU Canônica e são transparentes ao programador.

O "loop" de eventos é iniciado com a exibição do cursor para edição (91), operação periodicamente necessária. A seguir é feita a obtenção do código do evento (92). Esta obtenção de **Exemplo1** é **totalmente diferente** de **Exemplo**, pois o tratamento de eventos é feito em mais alto nível.

A identificação do evento continua sendo feita num grande CASE sobre os tipos de eventos (93). No entanto, os tipos de eventos são já direcionados aos objetos e suas atividades. A seguir, há a associação de um evento a seu tratamento pela aplicação (94-121). O fim do "loop" (122) ocorre quando a variável de controle de loop (47) tem seu valor TRUE, refletindo que o usuário escolheu a opção **Quit** do menu **File** (59).

Ao programador, a princípio, pode ficar transparente se o evento é causado por "mouse", teclado ou mesmo pela própria FIU.

O tratamento dos eventos é explicado a seguir.

Se o "click" do "mouse" foi dado fora da janela da aplicação, o tratamento (ativação de janela de DA) é feito já no manipulador de eventos (ver **Qual_Evento** no Anexo D).

Se um "click" do "mouse" foi dado na barra de menus ou uma tecla de comando é acionada diretamente, o evento é SELECAO_MENU (94). A apresentação do menu correspondente e o controle da seleção de itens é feito pelo manipulador de eventos, que identifica qual item de qual menu de qual janela foi selecionado. A execução da ação associada é feita pela rotina **DoCommand** (95).

Se o "click" do "mouse" foi dado na barra de títulos da janela, conhecida por "drag region", o evento é

MOVER_JANELA (96). A ação associada movimenta a janela (97).

Se o "click" foi dado na região de conteúdo da janela, o manipulador de eventos identifica que tipo de evento ocorreu: seleção de janela, SELECAO_JANELA (98), seleção de texto, SELECAO_TEXTO (102), ativação de janela, ATIVAR_JANELA(105), ou desativação de janela , DESATIVAR_JANELA(110) e a ação apropriada é tomada.

Se o evento é causado por teclas pressionadas , o manipulador de eventos testa se estas teclas formam um código de função (ALT-P ou CTRL-Q, por exemplo) para seleção de menu (tratada em SELECAO_MENU) ou se são teclas comuns para entrada de texto, causando o evento EDICAO_TEXTO (100).

Se o evento é um evento de atualização, ATUALIZAR_JANELA (115), significa que a visão da janela deve ser atualizada . O trecho de atualização (no caso, apenas a atualização do texto exibido na janela) é tipicamente limitado por comandos de início (117) e fim (119) de atualização.

9 ENFOQUES DE TRADUÇÃO DA FIU CANÔNICA

Neste capítulo são descritos dois enfoques de tradução da FIU Canônica para as FIUs subjacentes (no caso Mac Toolbox e MS-Windows).

9.1 Introdução

Como visto, a FIU Canônica é uma ferramenta usada para prover suporte ao projetista de interface e ao programador de aplicação. Entretanto, mais que implementar a FIU Canônica, com a execução de todas suas tarefas desde as mais primitivas até as mais complexas, optou-se pela descrição do processo de tradução (ou ainda de correspondência ou mapeamento) entre os objetos/operações da FIU Canônica e os objetos/operações das FIUs subjacentes (no caso, a Mac Toolbox e o MS-Windows). Esta opção foi tomada por três motivos principais:

- a preocupação do trabalho é com o diálogo interno e não com o diálogo externo ("look and feel"), levando à priorização da uniformização da comunicação IU-aplicação;

- a flexibilidade decorrente do uso possível de vários diálogos externos, desde que a tradução da FIU Canônica para as FIUs que os suportam seja provida;

- facilidade de experimentação, pois é mais direto construir tradutores do que FIUs.

A FIU Canônica funciona então como uma **camada intermediária entre a aplicação e as ferramentas disponíveis**, abstraindo as idiossincrasias de cada uma. No processo de tradução, os elementos estáticos e dinâmicos da FIU Canônica devem ser mapeados para os elementos correspondentes das FIUs subjacentes específicas. A tradução pode ser feita segundo dois enfoques principais: o enfoque compilativo e o enfoque interpretativo.

No primeiro enfoque as chamadas à FIU Canônica são compiladas (daí o nome **compilativo**) gerando chamadas a FIU específica. No segundo enfoque, um reconhecedor incremental interpreta (daí o nome **interpretativo**) as ações do usuário e, baseado na descrição da IU, ativa as rotinas da aplicação e as chamadas a FIU específica.

Cada um dos enfoques parte da existência de uma definição de IU e de uma definição e estruturação da aplicação. A definição de IU é tipicamente feita pelo Projetista de Interface, enquanto a aplicação é tipicamente definida por um Analista da Aplicação e implementada por um Programador de Aplicação.

A interação entre a aplicação e a IU (diálogo interno) é estabelecida de acordo com tipo de controle da interação. Se a IU é ativada sob controle da aplicação (**controle interno**), a FIU deve ser organizada como uma biblioteca de rotinas de interação ("**toolbox**"). Se a IU ativa a aplicação (**controle externo**), esta é que deve ser organizada como uma biblioteca de ações semânticas, e à FIU cabe toda a responsabilidade de tratar as ações do usuário e chamar adequadamente as rotinas da aplicação. Isto acontece se a FIU Canônica for uma "**toolkit**".

Em verdade, o enfoque de tradução está intimamente relacionado com o tipo de controle de interação adotado pela FIU Canônica. Preferencialmente, o enfoque compilativo é adotado pois a FIU Canônica é uma "toolbox". Apesar do interpretativo ser indicado apenas se a FIU Canônica for uma "toolkit", sua descrição também é feita.

Uma descrição dos processos de tradução da FIU Canônica segundo cada um dos enfoques é dada a seguir. Uma versão preliminar desta descrição está contida em [FPP 90].

9.2 Enfoque Compilativo

O enfoque compilativo pode ser usado quando o programa detém o controle da interação, isto é, quando o tipo de controle é **interno**. Esta estrutura de controle é típica para "toolboxes", conforme capítulo 3. Se a FIU Canônica configurar-se como uma "toolbox", então o enfoque compilativo é mais adequado para a tradução.

A figura 9.1 apresenta um esquema do processo de tradução segundo este enfoque.

Primeiramente, o **Projetista de Interface** especifica a IU associada ao programa usando o MRIU da FIU Canônica (**Especificação da Interface usando Canonicus**). O **Programador de Aplicação** desenvolve o programa interativo (definido pelo **Analista** de aplicação) inserindo, nos pontos do programa em que a IU deve ser acionada, chamadas à FIU Canônica, ou seja, usando o protocolo definido para os objetos de IU da FIU Canônica. Este protocolo por

construção é independente de FIU subjacente utilizada, tornando a aplicação portável.

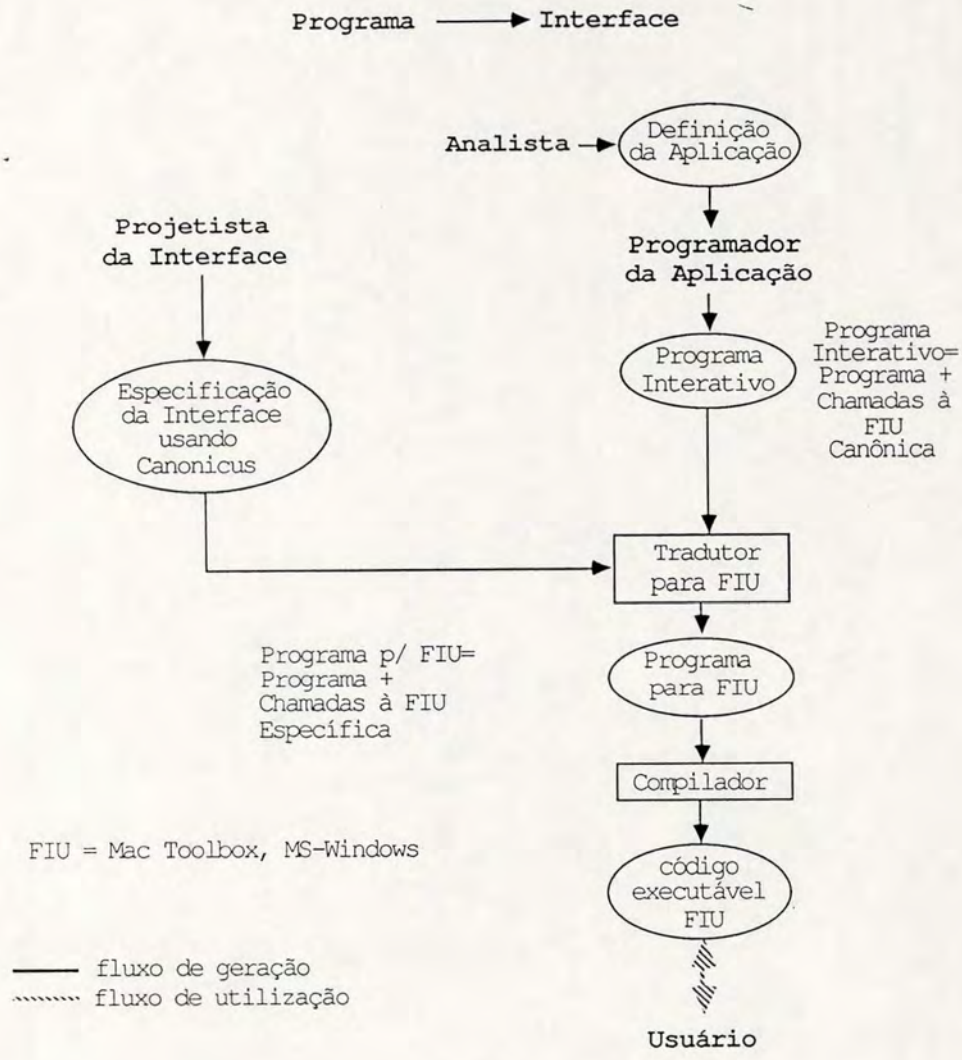

Figura 9.1 - Enfoque Compilativo

Para a execução do programa interativo, o protocolo genérico da FIU Canônica deve ser traduzido para rotinas de interação de um ambiente específico, substituindo-se qualquer referência a elementos da FIU Canônica por uma referência a elementos presentes na FIU

específica. Esta tradução é feita por um tradutor para FIU e gera um programa interativo para uma FIU determinada (**Programa para FIU**) com as devidas definições, chamadas ou corpos de procedimentos necessários para que a FIU execute, sem perda de poder, a IU especificada através da FIU Canônica.

O tradutor responsável pelo mapeamento do programa portável (**Programa Interativo**) para um programa que utiliza uma FIU determinada pode ser desenvolvido através de geradores de reconhecedores como o YACC [JOH 75] ou PL [FRA 89], uma ferramenta disponível no projeto AMADEUS [PRI 87].

Como as chamadas de rotinas de IU, tanto da FIU Canônica quanto da FIU específica, estão embutidas no programa e este é desenvolvido numa linguagem de programação (como Pascal ou C) o passo seguinte é submeter o programa gerado (**programa para FIU**) ao **compilador** disponível no ambiente para que se obtenha o código executável para o programa (**código executável FIU**). Através da execução deste código, o usuário final interage com o programa interativo.

O enfoque compilativo realiza todo o mapeamento **a tempo de compilação**, ou seja, nenhuma tradução é feita a tempo de execução. Como em toda "toolbox", a amarração entre as rotinas da FIU Canônica e a aplicação faz-se através do embutimento das primeiras na segunda. As interações ocorrem nos pontos previstos do programa.

Um exemplo de uso deste enfoque para a tradução de uma linguagem genérica de definição de interface com o usuário (IDL) para uma específica (GIM-CPGCC/UFRGS) está detalhadamente descrito em [FPP 90] e foi um dos experimentos que comprovou a viabilidade do uso deste enfoque.

9.2.1 Roteiro para implementação de um tradutor

Um tradutor que use o enfoque compilativo de tradução da FIU Canônica é tipicamente um **pré-processador.** Sua função é "varrer" o código fonte do programa interativo e realizar o mapeamento descrito anteriormente.

Esta função é simples apenas na aparência pois implicitamente o que se está fazendo é a tradução de trechos de código desenvolvidos segundo o PDOO para trechos de código expressos no paradigma convencional de rotinas e funções.

A partir da especificação das classes genéricas e concretas para uma FIU específica (consultando a **Especificação da Interface usando Canonicus,** conforme figura 1), o pré-processador deve realizar as seguintes tarefas:

1) Inserir as declarações de variáveis e tipos específicos necessários à execução das rotinas da FIU específica;

2) Substituir as chamadas/invocações das operações/rotinas da FIU Canônica pelos trechos de código relativos à implementação destas rotinas na FIU particular.

Para descrição dos procedimentos do pré-processador, considere-se a seguinte estrutura de definição de uma classe:

```
Concret Class <nome_classe>
Inherit <super_classe>
Export
     <protocolo>
Feature
     <atributo> : <tipo>; -- declarações
     <atributo> : <tipo>; -- de atributo
     ...
     <rotina> ( <conj_de_parametros> ) is
     local <atributo> : <tipo>
     do
          <corpo de rotina>
     end
```

e o seguinte corpo de programa interativo (em Pascal):

```
PROGRAM <nome_programa>
USE <lista_de_classes_usadas>;

<lista_de_declarações_de_procedures>
<lista_de_declarações_de_variaveis>

BEGIN

     <comando>       /* comandos     */
     <comando>       /*    da        */
     <comando>       /* aplicação    */

     <ident>.<rotina> ( <parametros> );
          /* invocação de rotina   */
          /* da FIU Canônica       */

     <comando>       /* comandos     */
     <comando>       /*    da        */
     <comando>       /* aplicação    */
END.
```

O algoritmo para a tarefa 1 acima é o seguinte:

```
Para toda classe presente em <lista_de_classes_usadas>
        da cláusula USE do programa
Fazer
  Inicio
     Para toda declaração (local ou não) de atributos
               presente na cláusula Feature da
                definição da classe
     Fazer
       Início
       decl := declaração de atributo corrente
               no formato <atributo> : <tipo>;
       Se escopo  = local
       Então escopo  := normal;
       Conforme categoria do <tipo> do atributo
       Fazer
          Se categoria = Pré-definido
          Então Conforme <tipo>
                Fazer
                Se <tipo> = INTEGER
                Então <tipo> := (tipo correspondente
                                 na linguagem)
                Se <tipo> = CHAR
                Então <tipo> := (idem)
                Se <tipo> = BOOLEAN
                Então <tipo> := (idem)
                Se <tipo> = REAL
                Então <tipo> := (idem)
                Se <tipo> = STRING
                Então <tipo> := (idem)
                Fim /* conforme <tipo> */
          Se categoria = Tipo Classe
          Então Se a Classe existir
                Então anexa Classe na <lista_de_classes
                      _usadas>
                Senão mensagem de ERRO "Classe não
                      especificada"
          Se categoria = Tipo FIU
          Então Copia declaração do atributo para o
                código do programa sem alteração
       Fim /* conforme categoria */
       Se  declaracao decl já está presente na
           <lista_de_declarações_de_variaveis>
           do programa
       Então substituir a declaração antiga
             por decl ("overriden")
       Senão inserir declaração decl em
             <lista_de_declarações_de_variaveis>
             do programa
       Fim /* para toda declaração*/
  Fim /* para toda classe */
```

Este algoritmo faz com que todas **declarações de atributos das classes** de IU sejam traduzidas para **declarações de variáveis globais** do programa, pois:

- as estruturas de dados relativas à FIU nas "toolboxes" são tipicamente estruturas globais; e

- o tratamento de herança de atributos de objetos via níveis de escopo de linguagens bloco-estruturadas é complexo e está além do objetivo deste trabalho.

Os atributos locais (presentes na cláusula local da definição da classe), por exemplo, serão tratados como atributos normais (com qualquer uma das suas três categorias, a saber, **pré-definido, tipo Classe** e **tipo FIU**) e mesmo assim o algoritmo garante que será definido apenas uma vez globalmente apesar das repetidas definições locais.

O estado inicial (antes da execução do pré-processador) de **<lista_de_declarações_de_variaveis>** é o conjunto de declarações das variáveis do programa e o seu estado final (após a execução do pré-processador) é o conjunto resultante da união das declarações das variáveis do programa com as declarações das variáveis e estruturas de dados para uso das FIUs específicas.

O pré-processador para uma FIU específica deve conhecer:

- os tipos pré-definidos do Canonicus e da linguagem em que o programa está desenvolvido (ver categoria **Pré-definido** no algoritmo);

- as classes definidas para a FIU Canônica, a partir da especificação da IU (ver categoria **Tipo Classe** no algoritmo); e

- os tipos de estruturas de dados para a FIU específica (ver categoria **Tipo FIU** no algoritmo).

O algoritmo para a tarefa 2 é o seguinte:

```
Para toda invocação de rotina da FIU Canônica
     com formato <ident>.<rotina> ( <parametros> )
     presente na posição <pos> no corpo do programa
Fazer
     Conforme <rotina>
     Fazer
          Se <rotina> = "Criar"
          Então Inserir em <pos>:
                - código para inicialização dos
                  atributos da classe <ident> e
                  de suas subclasses genéricas e
                  concretas; os valores iniciais,
                  se não houver inicialização explícita
                  na definição da classe, serão
                  os valores "default";
                - trecho para execução dos procedi-
                  mentos da rotina "Criar" da classe
                  concreta para a FIU específica
          Se <rotina> = "Qual_evento"
          Então Início
                 Inserir em <lista_de_declarações
                   _de_procedures> a declaração da função
                   "Qual-evento" com este mesmo nome;
                 Inserir em <pos> a invocação da
                   função "Qual_evento" atribuindo
                   valor de retorno a <ident>, que deve
                   ser um objeto da classe Manipulador_
                   _de_Eventos previamente criado
               Fim
```

```
            Se <rotina> = Qualquer outra
            Então Se <ident> é um objeto já criado
                  Então Inserir em <pos> o trecho para
                        execução dos procedimentos da
                        rotina presente na classe
                        concreta para a FIU específica
                  Senão Mensagem de Erro "Referência a objeto
                        Inexistente"
      Fim /* conforme <rotina> */
Fim /* para toda invocação */
```

Tipicamente a posição **<pos>** é o local da invocação da rotina da FIU Canônica no corpo do programa. Além disto, o trecho para execução dos procedimentos de cada rotina é o trecho de código compreendido entre as palavras reservadas **do** e **end**, ou seja, o **<corpo de rotina>** na definição da classe.

Eventualmente, o pré-processador precisa realizar alguns procedimentos adicionais. Para o Mac Toolbox, por exemplo, as inicializações relativas aos subgerentes, necessárias para o correto uso das rotinas da "toolbox", devem ser inseridas no início do programa na ordem especificada em /APP 85/.

9.3 Enfoque Interpretativo

O enfoque interpretativo pode ser usado quando a IU detém o controle da interação, isto é, quando o tipo de controle é **externo**. Esta estrutura de controle é típica para "toolkits", conforme capítulo 3. Se a FIU Canônica configurar-se como uma "toolkit", então o enfoque interpretativo é mais adequado para a tradução.

A figura 9.2 apresenta um esquema do processo de tradução segundo este enfoque.

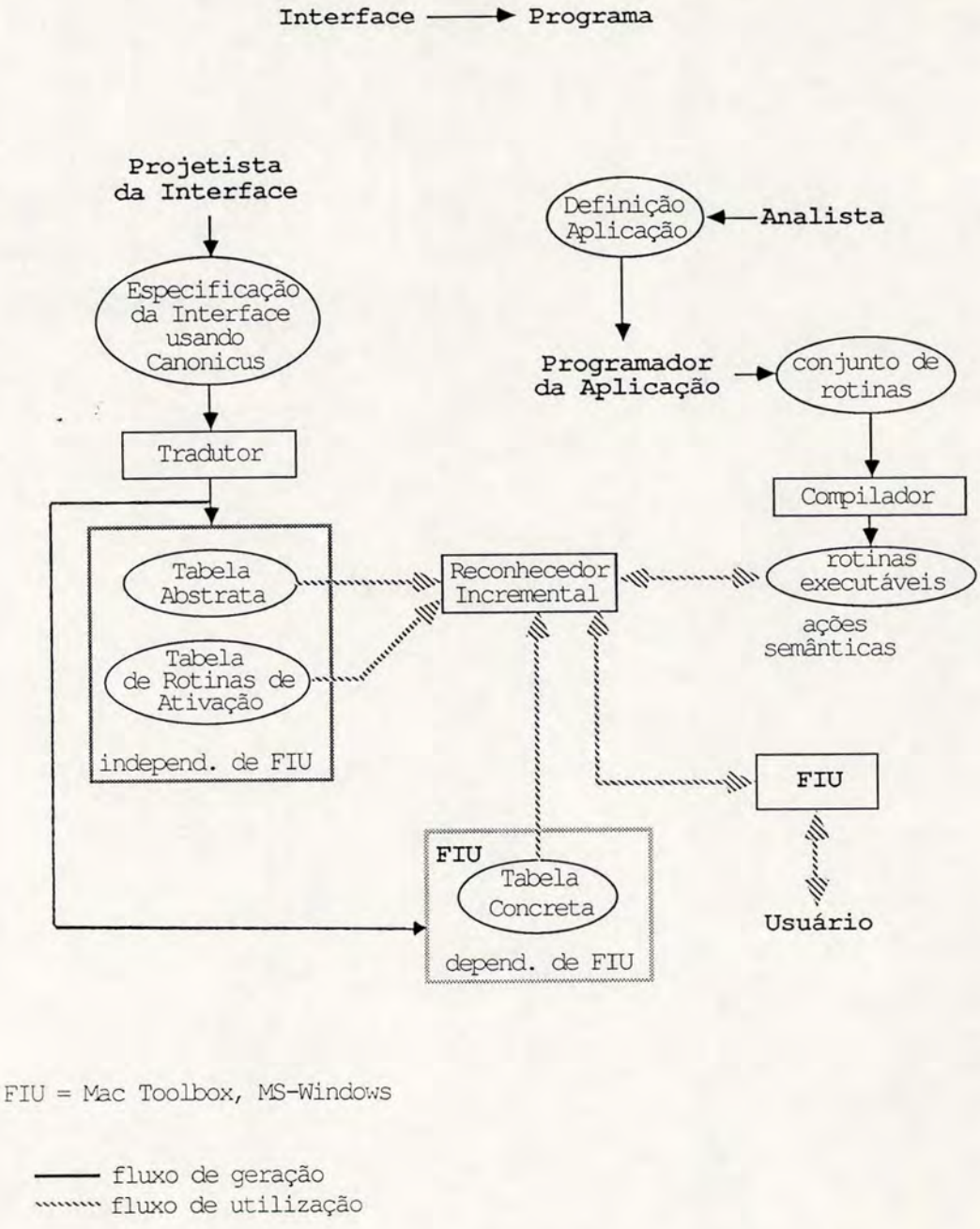

Figura 9.2 - Enfoque Interpretativo

A especificação da IU (**Especificação da Interface usando Canonicus**), feita pelo **Projetista de Interface**, agrega além da descrição dos aspectos relativos à IU, a associação a ações semânticas da aplicação, ou seja, a ativação das rotinas da aplicação e os parâmetros necessários para sua execução. Este é o procedimento

costumeiro no uso de uma "toolkit" onde o projetista de interface combina os elementos de IU e define lacunas onde serão encaixadas as ações da aplicação, na forma de rotinas executáveis.

O **Programador de Aplicação** desenvolve as rotinas da aplicação (definidas pelo **Analista**) na forma de procedimentos e ou funções (**conjunto de rotinas**) que, compiladas, dão origem às **rotinas executáveis** que são conectadas a IU. Estas rotinas são também denominadas **ações semânticas** do programa e formam uma biblioteca de ações da aplicação. Para formar estas bibliotecas, o ambiente de programação deve possuir características como definição de rotinas em unidades e compilação separada das unidades.

O processo de interpretação é feito por um **reconhecedor incremental** e é totalmente dirigido por tabelas. As tabelas são geradas a tempo de compilação e o reconhecedor, baseado nas informações contidas nas tabelas, realiza a tradução a tempo de execução.

As tabelas usadas são:

- tabela concreta
- tabela abstrata
- tabela de rotinas da ativação

As tabelas abstrata e concreta são geradas a partir da especificação da IU. Um **tradutor** (possivelmente

desenvolvido através de geradores de tradutores) monta a tabela abstrata, a tabela de rotinas de ativação e uma tabela concreta.

A **tabela concreta** contém elementos/operações reais disponíveis em FIUs existentes, baseada nas equivalências definidas nas classes concretas. Por isto, é denominada **dependente de FIU.** O reconhecedor mapeia os objetos/operações concretos da FIU Canônica para objetos correspondentes em uma FIU específica. A tabela concreta envolve as informações presentes nas descrições das classes concretas, ou seja, implementação dos objetos da FIU Canônica através de elementos e operações de FIUs reais existentes, definindo a apresentação da IU.

A **tabela abstrata** envolve as informações presentes nas descrições das classes abstratas da FIU Canônica, ou seja, conexões com a aplicação e interrelações existentes entre os componentes da IU (hierarquia de objetos componentes, protocolos, etc).

A **tabela de rotinas de ativação** contém a associação da tabela abstrata da IU às rotinas da aplicação, baseada na associação descrita na especificação da IU.

Estas duas últimas tabelas, lidando no nível abstrato, são denominadas **independentes de FIU.**

O **reconhecedor incremental**, dirigindo-se por estas tabelas, gerencia a IU para ativar a aplicação. Uma

ação do usuário gera eventos de uma FIU específica que são detectados pelo reconhecedor incremental via tabela concreta. Como, por construção da FIU Canônica, a cada operação concreta significativa para o programa corresponde uma abstrata, o reconhecedor consulta a tabela abstrata. Com a informação de qual operação abstrata foi feita, o reconhecedor consulta a tabela de rotinas de ativação, e obtém a identificação da rotina da aplicação que deve ser chamada para atender a solicitação do usuário.

Num processo semelhante a um reconhecimento incremental em editores dirigidos por sintaxe, as ações da aplicação são ativadas quando uma ação sobre um objeto de IU é reconhecida, sendo então disparada, via tabela abstrata, a rotina executável correspondente. O reconhecedor incremental é obtido via gerador de reconhecedores incrementais, como por exemplo o disponível no projeto AMADEUS [ESP 89].

Neste enfoque, a cada ação do usuário (ou série de ações) na IU, uma ou mais rotinas da FIU são acionadas. Quando alguma computação é necessária, o componente computacional, na forma de **rotinas executáveis**, é chamado e lhe são passados os parâmetros recebidos do usuário.

Este enfoque baseia-se na possibilidade de tratar algumas decisões de interação **a tempo de execução** e algumas **a tempo de compilação**, em contraste com o enfoque anterior, que trata tudo a tempo de compilação.

9.4 Observações

Para tornar mais clara a diferença entre os dois enfoques, pode-se pensar que, no enfoque compilativo, as rotinas da FIU que lidam com a IU estão presentes numa biblioteca, sendo acessadas pelo programa. Já no enfoque interpretativo, as ações da aplicação é que estão em uma biblioteca, sendo acessadas pela interface. A figura 9.3 procura comparar estas duas abordagens.

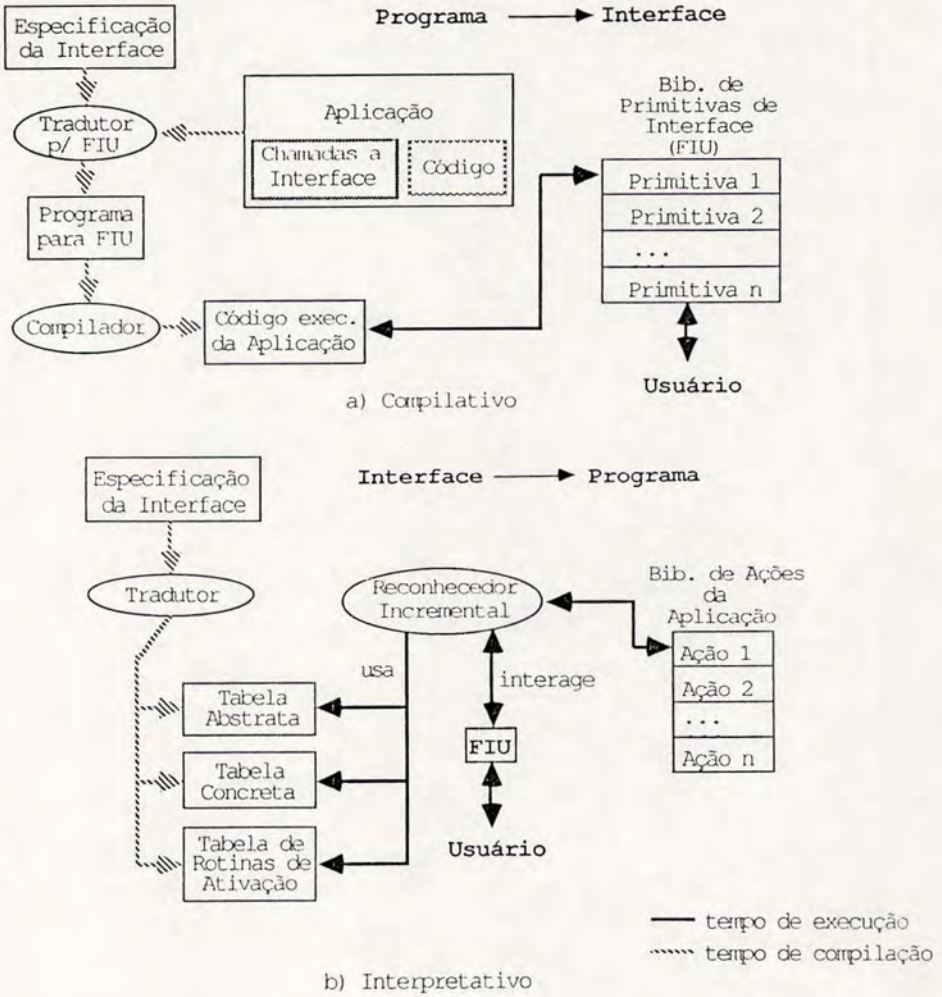

Figura 9.3 - Comparação dos enfoques

Ambos os enfoques de tradução propiciam a **independência de diálogo**, facilitando a modificação da descrição da IU sem necessidade de modificação na aplicação. No enfoque interpretativo, porém, esta característica é ainda mais forte pois as rotinas de aplicação, os objetos da IU e sua apresentação podem ser modificados não só independentemente mas também dinamiçamente, isto é, a tempo de execução.

No entanto, o enfoque interpretativo requer a construção de facilidades de IU distintas das atualmente disponíveis nas FIUs comerciais. FIUs comerciais não permitem acoplamento a um reconhecedor incremental e não possuem um formato executável imediato. Sua execução dá-se apenas através de rotinas embutidas em programas de aplicação.

Estima-se que o enfoque compilativo pode ser mais eficiente em termos de execução, já que não há ações (de consulta ou mapeamento) a serem executadas durante a interação. Além disto, pode ser mais facilmente implementado pelas características das FIUs disponíveis, enquanto o enfoque interpretativo pode ser mais conveniente em termos de flexibilidade de modificação da IU e da aplicação.

10 CONCLUSÕES

"There isn't any there there"
Gertrude Stein

Neste trabalho foi proposta a FIU Canônica, que visa diminuir a dificuldade de uso das FIUs comerciais (em particular Mac Toolbox e MS Windows) e propiciar a portabilidade de programas interativos desenvolvidos sobre elas.

A FIU Canônica é uma camada padronizada entre estas e a aplicação e é implementada através de mecanismos de tradução, com seus objetos e operações mapeados para objetos e operações das FIUs subjacentes.

Com o uso da FIU Canônica, a substituição de uma FIU subjacente por outra é **transparente** à aplicação. Na prática, a parte da aplicação que deve ser mudada é a parte associada ao controle, ou seja, a seqüência em que as rotinas da FIU são chamadas. Como a FIU Canônica (e também as FIUs Mac Toolbox e MS Windows) é uma "toolbox", o controle está na aplicação (controle interno). Para tornar a aplicação totalmente independente de FIU, o controle deve ficar todo com a IU (controle externo) e a aplicação deve ser organizada apenas como um conjunto de rotinas semânticas. Isto acarreta duas alterações significativas:

a) a FIU Canônica deixa de ser uma "toolbox" tornando-se uma "toolkit" (pois não possui as funções necessárias para se configurar um SGIU) e

b) uma mudança radical na estrutura dos programas, que passam a ser um conjunto de funções semânticas.

A arquitetura da FIU Canônica e seu modelo de representação Canonicus são desenvolvidos segundo o paradigma de orientação a objetos. Além disto, o Canonicus possui abstrações que provêm um suporte mais adequado aos usuários projetistas, mais notadamente o programador de aplicação.

A FIU Canônica serve como suporte a uma metodologia de desenvolvimento de IU, com a separação dos papéis do usuário projetista e com sua hierarquia de classes propiciando projeto de IUs com níveis diferenciados de detalhes. Canonicus é uma notação para projeto e implementação de IUs, assim como Eiffel é notação tanto para projeto quanto para implementação de aplicações. Os mesmos conceitos e construções são usados em ambos os estágios, com diferenças apenas no nível de abstração e nos detalhes relevantes a cada um. O uso do **mesmo formalismo** para ambas atividades evita os erros decorrentes da tradução de um formalismo de projeto para um de implementação e beneficia a tarefa do projetista de interfaces.

O programador de aplicação, por sua vez, pode usar as operações da FIU Canônica para invocar funções de IU de modo mais abstrato, sem se preocupar em conhecer ou fornecer os detalhes para a sua execução.

A FIU Canônica não pretende ser um marco na pesquisa sobre FIUs mas é uma contribuição no sentido de

facilitar o trabalho daqueles que desenvolvem programas interativos.

Como tarefa futura imediata, destaca-se a implementação da FIU Canônica, desenvolvendo os tradutores necessários.

A continuidade desta pesquisa pode-se desenvolver ainda em, no mínimo, duas direções, não mutuamente exclusivas:

i) a investigação do X-Windows (X-LIb, X-Toolkit, X-Intrinsics) para propiciar o uso da FIU Canônica também para aplicações no ambiente UNIX;

ii) a evolução visando um "toolkit" Canônico, incluindo as modificações previstas nesta dissertação.

ANEXO A ELEMENTOS, OPERAÇÕES E EVENTOS DO MAC TOOLBOX

Observações:

- Os elementos, suas propriedades e relacionamentos são apresentados como registros, na notação Pascal usada pela Mac Toolbox.

- As operações são agrupadas sucessivamente por objetos sobre os quais agem e pelo tipo de função que realizam. Mais detalhes (parâmetros e retornos, por exemplo) são encontrados em /APP 85/.

- Os eventos são apresentados por sua estrutura e seus tipos. A estrutura de um evento é dada também como registro Pascal. Os tipos de eventos são listados como constantes com valores associados.

1) Elementos do Mac e suas propriedades:

```
Janela
WindowRecord = RECORD
   port: GrafPort;              -- porta gráfica
   windowkind:INTEGER;          -- classe
   visible: BOOLEAN;            -- TRUE se visível
   hilited:BOOLEAN;             -- TRUE se realçada
   goAwayFlag:BOOLEAN;
               -- TRUE se tem região goAway
   structRgn:RgnHandle;         -- região de estrutura
   contRgn:RgnHandle;           -- região de conteúdo
   updateRgn:RgnHandle;         -- região de atualiz.
   windowDefProc:Handle;        -- função de definição
   dataHandle:Handle;
               -- usado pela windowDefProc
   titleHandle:StringHandle;-- titulo
   titleWidth:INTEGER;          -- tamanho do titulo
   controlList:ControlHandle;-- lista de controles
                                -- aponta o primeiro
   nextWindow: WindowPeek;      -- proxima janela
   windowPic:PicHandle;
               -- Picture para desenhar a janela
   refCon: LONGINT;             -- valor de referência
   END;
```

```
Caixa de diálogo:
DialogRecord = RECORD
   window: WindowRecord;      -- janela de dialogo
   items: Handle;             -- lista de itens
   textH: TEHandle;           -- texto editável corrente
   editField: INTEGER;        -- posicao que antecede
                              -- texto editavel
   aDefItem: INTEGER;         -- item default
   END;


DialogTemplate = RECORD
   boundsRect : Rect;         -- retângulo
   procId: INTEGER;           -- Id da defin. janela
   visible : BOOLEAN;         -- TRUE se visível
   goAwayFlag:BOOLEAN;
               -- TRUE se tem regiao goaway
   refCon : LONGINT;
               -- valor de referencia à janela
   itemsId : INTEGER;         -- Id do recurso dos itens
   title : Str255;            -- titulo
   END;


Menu:
MenuInfo = RECORD
   menuId: INTEGER;           -- identificador
   menuWidth: INTEGER;        -- largura
   menuHeight: INTEGER;       -- altura
   menuProc: Handle;          -- proc. de definição
   enableflags: LONGINT;
         -- flags de habilitacao de itens
   menuData: Str255           -- titulos e outros dados
   END;


Controles:
ControlRecord = RECORD
   nextControl: ControlHandle;-- próx. controle
   contrlOwner: WindowPtr;    -- janela que o contém
   contrlRect: Rect;          -- retângulo
   contrlVis: Byte;           -- 255 se visivel;
   contrlHilite : Byte;       -- intensidade de realce
   contrlValue: INTEGER;      -- valor corrente
   contrlMin: INTEGER;        -- valor mínimo
   contrlMax : INTEGER;       -- valor máximo
   contrlDefProc : Handle;    -- função de defin.
   contrlData: Handle;        -- dados para
                              -- contrlDefProc
   contrlAction: ProcPtr;     -- ação default
   contrlRfCon: LONGINT;      -- valor de referência
   contrlTitle : Str255;      -- título
   END;
```

```
Texto:
TERec = RECORD
   destRect: Rect;            -- retângulo destino
   viewRect: Rect;            -- retângulo de visão
   lineHeight: Rect;          -- usado para espaçamento
   selStart:INTEGER;
     -- início de intervalo de seleção
   selEnd: INTEGER;
     -- fim de intervalo de seleção
   just: INTEGER;             -- alinhamento
   teLenght: INTEGER;         -- tamanho
   hText: Handle;             -- texto a ser editado
   txFont: INTEGER;           -- fonte
   txFace: Style;             -- estilo do caracter
   txSize: INTEGER;           -- tamanho do caracter
   inPort: GrafPtr;           -- porta gráfica
   nLines: INTEGER;           -- numero de linhas
   lineStarts: Array[0 .. 16000] OF INTEGER;
                      -- posicoes de inicio de linha
   END;
```

2) Operações

A) Sobre Janelas

Inicialização e alocação:
InitWindows, GetWMgrPort, NewWindows,
GetNewWindow, CloseWindow, DisposeWindow

Exibição:
SetWTitle, GetWTitle, SelectWindow,
HideWindow, ShowWindow, ShowHide,
HiliteWindow, BringToFront,
SendBehind, FrontWindow, DrawGrowIcon

Localização do Mouse:
FindWindow, TrackGoAway

Mudança de Janelas:
MoveWindow, DragWindow, GrowWindow,
SizeWindow

Manutenção da Região de Atualização:
InValRect, InValRgn,ValidRect,ValidRgn,
BeginUpdate, EndUpdate

Miscelaneas:
SetWRefCon, GetWRefCon, SetWindowPic,
GetWindowPic, PinRect, DrawGrayRgn

B) Sobre Controles

Inicialização e Alocação:
NewControl, GetNewControl, DisposeControl,
KillControls

Exibição:
SetCTitle, GetCTitle, HideControl,
ShowControl, DrawControls, HiliteControl
Localização do Mouse:
FindControl, TrackControl, TestControl
Mudança de Controle:
MoveControl, DragControl, SizeControl
Atribuição de Valores:
SetCtlValue, GetCtlValue, SetCtlMin,
GetCtlMin, SetCtlMax, GetCtlMax
Miscelaneas:
SetRefCon, GetRefCon, SetCtlAction,
GetCtlAction

C) Sobre Menus

Inicialização e Alocação:
InitMenus, NewMenu, GetMenu, DisposeMenu
Formação de Menus:
AppendMenu, AddResMenu, InsertResMenu
Formação da Barra de Menus:
InsertMenu, DrawMenuBar, DeleteMenu,
ClearMenuBar,GetNewMBar, GetMenuBar,
SetMenuBar
Escolha de Menu:
MenuSelect, MenuKey, HiliteMenu
Controle de Aparição de Itens:
SetItem, GetItem, DisableItem,
EnableItem,CheckItem, SetItemMark,
GetItemMark, SetItemIcon, GetItemIcon,
SetItemStyle, GetItemStyle
Miscelâneas:
CalcMenuSize, CoutMItens, GetMHandle,
FlashMenuBar, SetMenuFlash

D) Sobre Texto

Inicialização e Alocação:
TEInit, TENew, TEDispose
Acesso a Texto de Registro de Edição:
TeSetText, TeGetText
Inserção e Seleção de Texto:
TEIdle, TEClick, TSetSelect,
TEActivate, TEDesactivate

```
Edição:
        TEKey, TECut, TECopy, TEPaste,
        TEDelete, TEInsert
Exibição de Texto e "Scrolling":
        TeSetJust, TEUpdate, TextBox, TEScroll
Manipulação de "Scraps" ("Clipboard"):
        TEFromScrap, TEToScrap, TEScrapHandle,
        TEGetScrapLen, TESetScrapLen
Rotinas Avançadas:
        SetWordBreak, SetClipLoop, TECalText
```

E) Sobre Caixa de Diálogo

```
Inicialização e Alocação:
        InitDialogs, ErrorSound, SetDAFont
Criação e Remoção:
        NewDialog, GetNewDialog, CloseDialog,
        DisposDialog, CouldDialog, FreeDialog
Manipulação de Eventos:
        ModalDialog, IsDialogEvent, DialogSelect,
        DlgCut, DlgCopy, DlgPaste, DlgDelete,
        DrawDialog
Invocação de Alertas:
        Alert, StopAlert, NoteAlert,
        CautionAlert, CouldAlert, FreeAlert
Manipulação de Itens:
        ParamText, GetDItem, SetDItem,
        GetIText, SetIText, SelIText,
        GetAlrtStage, ResetAlrtStage
```

3) Eventos do Mac ToolBox

```
Evento:
EventRecord = RECORD
   what      : INTEGER;  /* tipo */
   message   : LONGINT;  /* mensagem */
   when      : LONGINT;
               /* tempo desde sinalização do evento */
   where     : Point;    /* coordenadas do mouse */
   modifiers : INTEGER;  /* "flags" modificadores */

Tipos:

CONST
   nulo       = 0  /* nulo */
   Mouse_down = 1  /* botão do mouse pressionado */
   Mouse_Up   = 2  /* botão do mouse liberado */
   Key_Down   = 3  /* tecla pressionada */
   Key_Up     = 4  /* tecla liberada */
   Auto_Key   = 5  /* tecla repetida */
   Update_Evt = 6  /* atualização de janela */
```

```
DiskEvt     = 7  /* inserção de  disco */
Activate    = 8  /* ativ. e desativ.  de janela */
AbortEvt    = 9  /* Opção "Cancel" */
NetWorkEvt = 10 /* para uso com AppleTalk */
DriverEvt   = 11 /* erro de dispositivo */
Applevt     = 12 /* definido pela aplicação */
App2evt     = 13 /* idem */
App3evt     = 14 /* idem */
App4evt     = 15 /* idem */
```

Posições (em janelas):

```
0 - InDesk        /* janela de um DA */
1 - InMenuBar     /* na barra de título */
2 - InSysWindow   /* na janela do sistema */
3 - InContent     /* dentro de uma janela */
4 - InDrag        /* na "drag region" duma janela*/
5 - InGrow        /* na "size box" duma janela */
6 - InGoAway      /* na "close box" duma janela */
```

ANEXO B ELEMENTOS, OPERAÇÕES E EVENTOS DO MS-WINDOWS

Observações:

- Alguns elementos do MS Windows são apresentados através de estruturas de registros e outros através da estrutura usada nos arquivos de recursos para defini-los.

- As operações são agrupadas por objetos a que se aplicam e tipo de função. Mais detalhes (parâmetros e retornos, por exemplo) são encontrados em /JAM 87/.

- Os eventos do MS Windows são um tipo de mensagem. A estrutura de uma mesnagem e os códigos dos tipos de eventos são listados sem valor associado. Mais detalhes sobre os eventos são dados em /JAM 87/.

1) Elementos e suas propriedades:

Janela:

```
WNDCLASS = RECORD
   style: WORD;            -- estilo da classe
   lpfnWndProc:FARPROC;
     -- função que processa mensagens
     -- para a janela
   cbClsExtra:int;         -- normalmente 0
   cbWndExtra:int;         -- normalmente 0
   hInstance: HANDLE;      -- módulo da classe
   hCursor: HCURSOR;       -- cursor do aplicativo
   hIcon: HICON;           -- icone do aplicativo
   hbrBackground:HBRUSH;-- cor de fundo
   lpszMenuName:LPSTR;
     -- pointer para recurso que
     -- contem menu da classe
   lpszClassName:LPSTR;
     -- pointer para nome da classe da janela
   END;
```

```
CREATESTRUCT = RECORD

   lpCreateParams:LPSTR;
     -- pointer para parâmetros
     -- de criação de janela
   hInstance: HANDLE; -- módulo da classe
   hwndParent: HWND;  -- Janela Pai
   cy: int;           -- altura da janela
   cx: int;           -- largura da janela
   x: int;
     -- coord x do pt superior esquerdo
   y: int;
     -- coord y do pt superior esquerdo
   style: long;       -- estilo da classe
   lpszName: LPSTR;
     -- pointer para nome da janela
   lpszClass:LPSTR;
     -- pointer para nome da
     -- classe da janela
   END;
```

Caixa de diálogo:(a partir dos recursos)

```
   nome     -- nome da caixa de diálogo
   x        -- coord x do pt superior esquerdo
   y        -- coord y do pt superior esquerdo
   largura  -- largura da caixa de diálogo
   altura   -- altura da caixa de diálogo
   estilo   -- estilo da janela da caixa
   caption  -- texto associado a caixa
   menu     -- menu da caixa de diálogo
            -- (normalmente vazio)
   class    -- classe da caixa de diálogo
```

Menu: (a partir dos recursos)

```
   nome                  -- nome do menu
   Para cada item:
       string            -- texto para opções
       valor resultado   -- número associado a item
       opcoes            -- tipos de aparência:
                         -- CHECKED, GRAY, INACTIVE
```

Controles: (a partir dos recursos)

```
tipo_controle -- tipo do controle
string        -- texto associado ao controle
id_controle   -- Id do controle
x             -- coord x do pt superior
                 esquerdo do controle
y             -- coord y do pt superior
                 esquerdo do controle
largura       -- largura do controle
altura        -- altura do controle
estilo        -- estilo do controle
```

Recurso:

```
nome      -- id do recurso
tipo      -- tipo do recurso
```

2) Operações

A) Sobre Janelas

Inicialização e alocação:
CreateWindow,DestroyWindow, GetWindowLong,
GetWindowWord, SetWindowLong, SetWindowWord

Exibição:
ShowWindow, OpenIcon, CloseWindow,
IsIconic, IsWindowVisible, AnyPopUp

Mudança de Janelas:
MoveWindow, BringWindowToTop,
SetActiveWindow

Atributos da Janela:
SetWindowText, GetWindowText,
GetWindowTextLenght

Auxiiares:
GetParent, FindWindow, Scrolling

Manutenção da Região de Atualização:
GetWindowDC, GetDC, ReleaseDC,
BeginPaint, EndPaint, UpdateWindow,
GetUpdateRect, InvalidateRect,
InvalidateRgn, ValidateRect, ValidateRgn

Conversão de Coordenadas:
ClientToScreen, ScreeToClient,
WindowFromPoint, ChildWindowFrom,
SetWindowOrg, GetWindowOrg,
SetWindowExt, GetWindowExt,
OffsetWindowOrg, ScaleWindowExt

B) Sobre Controles

Ver Caixa de Diálogo

C) Sobre Menus

Inicialização e Alocação:
SetMenu, GetMenu, DestroyMenu,
GetSubMenu
Formação de Menus:
ChangeMenu
Formação da Barra de Menus:
DrawMenuBar, ChangeMenu
Escolha de Menu:
GetMenuString

Controle de Aparição de Itens:
CheckMenuItem, EnableMenuItem,
HiliteMenuItem
Alteração do Menu Sistema:
GetSystemMenu

D) Sobre Texto

Inicialização e Alocação:
SetFocus, CreateCaret, ShowCaret,
SetCaretPos, GetCaretPos, DestroyCaret
Exibição de Texto e "Scrolling":
TextOut, DrawText

Manipulação de "Scraps" ("Clipboard"):
OpenClipBoard, SetClipBoardData,
GetClipBoardData, CloseClipBoard

Acesso a Texto de Registro de Edição,
Inserção e Seleção de Texto,
Edição,
Rotinas Avançadas:

Executadas através do envio de mensagens da aplicação à janela de controle de edição e não através de rotinas (ver Classe Texto_MSW no Anexo E)

E) Sobre Caixa de Diálogo

Inicialização e Alocação:
CreateDialog, DialogBox, IsDialogMessage,
EndDialog

```
Diretorio:
        DlgDirList, DlgDirSelect
Manipulação de Itens:
        GetDlgItem, SetDlgItem, GetDlgItemInt,
        CheckDlgButton, IsDlgButtonChecked,
        CheckRadioButton, SendDlgItemMessage
Conversão de Coordenadas:
        MapDialogRect
```

3) Eventos do MS Windows

```
Teclado:
   WM_CHAR, WM_COMMAND, WM_DEADCHAR,
   WM_KEYDOWN, WM_KEYUP
"Mouse":
   WM_LBUTTON, WM_MBUTTON, WM_BUTTONDBLCKL,
   WM_LBUTTONUP, WM_MBUTTONDBLCKL, WM_MBUTTONUP,
   WM_MOUSEMOVE, WM_RBUTTON, WM_RBUTTONDBLCLK,
   WM_RBUTTONUP
Outros:
   WM_TIMER, WM_HSCROLL, WM_SCROLL, WM_NCACTIVATE,
   WM_NCCALCSIZE, WM_NCCREATE, WM_NCDESTROY,
   WM_NCCHITTEST, WM_NCLBUTTONDBLCLK,
   WM_NCLBUTTONDOWN, WM_NCLBUTTONUP,
   WM_NCLMBUTTONDBLCLK, WM_NCLMBUTTONDOWN,
   WM_NCLMBUTTONUP, WM_NCLRBUTTONDBLCLK,
   WM_NCLRBUTTONDOWN, WM_NCLRBUTTONUP,
   WM_NCMOUSEMOVE, WM_NCPAINT

Sistema:
   WM_SYSCHAR, WM_SYSCOMMAND, WM_SYSDEADCHAR,
   WM_SYSKEYDOWN, WM_SYSKEYUP
```

Estes códigos são passados dentro de uma estrutura MSG.

```
MSG:
   hwnd: HWND;    -- janela destino da mensagem
   message:WORD;  -- um dos códigos
   wParam:WORD;   -- parametros
   lParam:LONG;   -- parametros
   time:DWORD;    -- "timestamp" de recepção da
                     mensagem
   pt:POINT;      -- posicao do mouse em coordenadas
```

ANEXO C DEFINIÇÃO DAS CLASSES GENÉRICAS PRIMITIVAS DA FIU CANÔNICA

Observações:

- Neste anexo são descritas, usando Canonicus, as classes genéricas primitivas da FIU Canônica. Estas classes são pré-definidas e reconhecidas pelo tradutor. Novas classes podem ser criadas a partir delas (ver capítulos 6 e 7).

- Os tipos de eventos da FIU Canônica são também descritos. Uma pequena explicação de cada tipo de evento é dada, incluindo os atributos do manipulador de eventos que são afetados por cada tipo. Cada tipo corresponde a uma constante inteira (o valor não está especificado).

1) Classes Genéricas

```
Generic Class  Objeto_de_IU
Export

     Criar, Id

Feature

Id : INTEGER;          -- identificador do objeto

Criar() : INTEGER;  -- Cria Objeto e retorna Id


Generic Class  Objeto_de_Interacao
Inherits  Objeto_de_IU
Export

     Recurso, Visivel, Criar, Exibir,
     Ativar, Desativar, Exibir


Feature

Id : INTEGER;              -- identificador do objeto
Recurso : Id_recurso;      -- Identificador de recurso
                           -- onde está a def. do objeto
Ativo : BOOLEAN;           -- TRUE, se objeto ativo
Visivel: BOOLEAN;          -- TRUE, se objeto visivel

Exibir  ();
```

```
Generic Class  Janela
Inherit  Objeto_de_Interacao
Export
     Titulo, Classe, Jan_Pai, Jan_Filha, Prox_jan,
     Lista_Itens, Selecionada, Icone_Assoc,
     Realce, Criar, Remover, Exibir, Esconder,
     Selecionar, Des_Selecionar, Realçar, Mover,
     Crescer, Trocar_Pos, Inic_Atual, Fim_Atual

Feature

Título: Texto;
Classe : INTEGER;  -- constante com valor de classe
Jan_Pai : Janela;
Jan_Filha : Janela;
Prox_Jan : Janela;
Lista_Itens : Controle;
Icone_Assoc  :  BOOLEAN;  -- TRUE: associada a icone

Criar () : INTEGER;     -- retorna Id para janela

Criar ( recurso ): INTEGER;
                -- idem usando recurso
Remover ();

Exibir ();

Esconder ();

Selecionar () ;

Des_Selecionar ();

Realcar () ;

Mover ( coordx, coordy : INTEGER );

Mover (pt_mousex, pt_mousey: INTEGER);

Crescer ( coordx, coordy : INTEGER ) ;

Crescer ( pt_mousex, pt_mousey : INTEGER );

Trocar_Pos ( outra_janela : INTEGER ) ;
               -- Id de janela a trocar posicao

Inic_Atual () ;

Fim_Atual () ;
```

```
Generic Class  Caixa_de_Dialogo
Inherit  Objeto_de_Interacao
Export
     Tipo, Jan_dialogo, Tipo, Lista_Itens,
     Item_Ed_Corr, Botao_Default, Prox_Dialogo,
     Criar, Remover

Feature

Jan_dialogo : Janela;     -- janela que a contém
Tipo : INTEGER;           -- constante com valor
                          -- modal, não_modal, alerta
Lista_Itens : Controle;   -- primeiro da lista
                          -- dos controles da
                          -- caixa
Item_Ed_Corr : Controle;  -- Id do item de edicao
                          -- corrente
Botao_Default : Controle;-- Id do botao default
Prox_Dialogo : Caixa_de_Dialogo;
          -- proxima caixa de dialogo (se aninhadas)

Criar () : INTEGER;       -- Id da Caixa_de_Diálogo;

Remover () ;


Generic Class Cx_Dialogo_Modal
Inherit Caixa_de_Dialogo
Export

     Criar

Feature

Criar(): INTEGER;  -- cria Caixa de Diálogo Modal


Generic Class Alerta
Inherit Caixa_de_Dialogo
Export

     Alerta, Bip

Feature

Alerta (tipo_alerta:INTEGER);
Bip () ;  -- sinal sonoro
```

```
Generic Class Menu
Inherit Objeto_de_Interacao
Export

     Titulo, Tipo, Menu_Pai, Menu_Filho,
     Prox_Menu, Jan_Menu, Lista_Itens,
     Remover

Feature

Titulo : Texto;
Tipo :   INTEGER;    -- constante com valor
                     -- Horizontal, Vertical ou Barra
Menu_Pai : Menu;
                -- implementa hierarquia de menus
Menu_Filho : Menu;  -- idem
Prox_Menu : Menu;
                -- idem  (mesmo nível e mesmo pai)
Jan_Menu : Janela;
                -- janela em que o menu é exibido
Lista_Itens : Texto;
                -- Id do primeiro item da lista
                -- Itens sao textuais

Remover ();


Generic Class Barra_Menu
Inherit Menu
Export
     Criar, Exibir, Selecionar, Des_Selecionar,
     Ins_Item, Rem_Item, Rem_todos_itens,
     Habil_Item, Desabil_Item


Feature

Criar () : INTEGER;

Exibir () ;

Selecionar ();

Des_Selecionar ();

Ins_Item ( txt_item: STRING; item : INTEGER );
          -- posicao indicada por item

Rem_Item (item : INTEGER );

Rem_todos_itens ();

Desabil_Item (item : INTEGER);
```

```
Generic Class Menu_Popup
Inherit Menu
Export

     Criar, Exibir, Ins_Item, Rem_Item,
     Habil_Item, Desabil_Item, Obter_item
     Marcar_Item, Desmarcar_Item, Alter_Item,
     Obter_item


Feature

Criar ( nome_menu : STRING ) : INTEGER;

Criar () : INTEGER;

Exibir () ;

Ins_Item ( txt_item: STRING; item : INTEGER );
          -- posicao indicada por item

Ins_Item ( txt_item : STRING );
          -- como ultimo item do menu

Ins_Item ( tipo_recurso : STRING );

Rem_Item (item : INTEGER );

Habil_Item ( item : INTEGER);

Desabil_Item (item : INTEGER);

Marcar_Item (item : INTEGER);

Desmarcar_Item ( item: INTEGER);

Alter_Item ( item: INTEGER;  novo_txt: STRING);

Obter_item (item: INTEGER) : STRING;


Generic Class Controle
Inherit Objeto_de_Interacao
Export

     Jan_Exib, Txt_Controle, Tipo, Prox_controle,
     Eh_Default, Criar, Remover, Exibir,
     Esconder, Realçar, Mover, Crescer,
     Marcar, Desmarcar
```

```
Feature

Jan_Exib : Janela;
Txt_Controle : Texto;
Tipo : INTEGER;
     -- constante inteira com valor:
     -- ( checkBox, RadioButton, Botao,
          Txt_Edit, Txt_Fixo, Icone );
Prox_controle : Controle;
Eh_Default : BOOLEAN;

Criar (): INTEGER;

Criar ( recurso: STRING ) : INTEGER;

Remover ();

Exibir ();

Esconder () ;

Realçar ();

Mover ( coordx, coordy : INTEGER );

Mover (pt_mousex, pt_mousey: INTEGER);

Crescer ( coordx, coordy : INTEGER ) ;

Crescer ( pt_mousex, pt_mousey : INTEGER );

Marcar ();

Desmarcar ();


Generic Class Dials
Inherit Controle
Export
     Valor_Min, Valor_Max

Feature

Valor_Min : INTEGER;
Valor_Max : INTEGER;
```

```
Generic Class Texto
Inherit Objeto_de_Interacao
Export

     Pos_Sel_Inic, Pos_Sel_Fim, Comprim_Txt,
     Alinhamento, Criar, Remover, Alinha,
     Scroll, Exibe_Cursor, Selecionar,
     Ativar, Desativar, Atualizar, Ins_txt,
     Alt_txt, Rem_txt, Cop_txt, Corta_txt,
     Cola_txt, Exibir, Exibir, ParaClip,
     DeClip

Feature

Pos_Sel_Inic : INTEGER;

     -- Posicao de inicio de texto selecionado

Pos_Sel_Fim : INTEGER;
     -- Posicao de fim de texto selecionado
     -- Se fim-inicio=0 eh ponto de insercao, ou seja,
     -- o cursor eh exibido na posicao e caracteres
     -- podem ser inseridos

Comprim_Txt : INTEGER;

Alinhamento : INTEGER; -- constante com valor
                       -- direita, esquerda, centro

Criar () ;

Remover () ;

Alinha (tipo alinhamento);

Scroll ( tipo scroll) ;

Exibe_Cursor () ;

Selecionar ( pt_mousex, pt_mousey : INTEGER);

Selecionar( pos_iniç pos_fim);

Ativar ();

Desativar ();

Atualizar ();

Ins_txt ( txt : STRING );

Alt_txt ( novo_txt: STRING );

Rem_txt ();
```

```
Cop_txt ();     -- COPY

Corta_txt ();   -- CUT

Cola_txt ();    -- Paste

Exibir (texto: STRING; coordx, coordy: INTEGER);

Exibir();

ParaClip ();    -- escreve no Clipboard

DeClip();       -- lê do Clipboard


Generic Class Manipulador_de_Eventos
Inherit Objeto_de_IU
Export

     Agente,          -- usuario ou aplicacao
     tipo,            -- tipo de evento
     timestamp,       -- marca de ocorrencia do evento
     janela,          -- janela onde ocorreu o evento
     menu,            -- menu onde ocorreu evento
     item,            -- item onde ocorreu evento
                      -- dentro de menu ou caixa dialogo
     char,            -- char teclado
     posicaox,        -- coord x do ponto do click
                      -- ou inicio de selecao de texto
     posicaoy,        -- coord y do ponto do click
                      -- ou fim de selecao de texto
     controle,        -- controle onde ocorreu evento
     caixa_dialogo,   -- caixa onde ocorreu evento
     janela,          -- janela onde ocorreu evento
     Qual_evento,     -- indica qual evento ocorreu

Feature

Agente : INTEGER;    -- constante com valor
                     -- USUARIO ou APLICACAO
tipo : INTEGER;      -- constante com valor
                     -- do tipo de evento
timestamp: INTEGER;
janela:INTEGER;
menu : INTEGER;
item : INTEGER;
char : CHAR;
posicao : INTEGER;
controle : INTEGER;
caixa_dialogo: INTEGER;
Qual_evento () : INTEGER
          -- retorna tipo de evento e atribui valores
          -- aos atributos acima
```

2) Tipos de Eventos da FIU Canônica

Retornado no atributo **tipo** do objeto Manipulador de Eventos (leia observações no início do anexo):

```
NULO              -- nulo
                  -- não afeta nada
SELECAO_MENU      -- seleção de item em menu
                  -- afeta menu e item
MOVER_JANELA      -- mudar posicao de janela
                  -- afeta janela
CRESCER_JANELA    -- mudar tam de janela
                  -- afeta janela
FECHAR_JANELA     -- fechar janela
                  -- afeta janela
CLICK_CONTROLE    -- click do mouse s/ um controle
                  -- afeta controle
SELECAO_TEXTO     -- selecao de texto
                  -- afeta posicaox e posicaoy
SELECAO_JANELA    -- selecao de janel
                  -- afeta janela
CLICK_DIALOGO     -- click do mouse s/ cx dialogo
                  -- afeta caixa_dialogo
ED_TEXTO_DLG      -- edicao de texto dentro de
                  -- uma caixa de dialogo
                  -- afeta caixa_dialogo e item
EDICAO_TEXTO      -- edicao de texto
                  -- afeta janela
ATIVAR_JANELA     -- ativacao de janela
                  -- afeta janela
DESATIVAR_JANELA  -- desativacao de janela
                  -- afeta janela
ATUALIZAR_JANELA  -- atualizacao de janela
                  -- afeta janela
TECLA             -- tecla pressionada
                  -- afeta char
TECLA_SOLTA       -- tecla liberada
                  -- afeta char
CLICK_MOUSE       -- "mouse" pressionado
                  -- afeta posicaox e posicaoy
MOUSE_SOLTO       -- "mouse" liberado
                  -- afeta posicaox e posicaoy
SCROLL_HORIZ      -- "scroll" horizontal da janela
                  -- afeta janela
SCROLL_VERT       -- "scroll" vertical da janela
                  -- afeta janela
EVENTO_DISPOSITIVO  -- erro de dispositivo
                  -- não afeta nada
INS_DISCO         -- insercao de disco
                  -- nao afeta nada
EVENTO_APLIC      -- evento reservado a aplicação
                  -- definir o que afeta
```

```
Modificadores (podem aparecer junto com o evento)

     CLICK_DUPLO     --modificando evento relacionado
                     --ao mouse indicando click duplo

     BOTAO_MOUSE     --modificando evento relacionado
                     --ao mouse (se este tem mais de um
                     --botão, como o MS-Windows) com
                     --os botões numerados da esq.
                     --para a dir. a partir de 1
```

ANEXO D CLASSES CONCRETAS PARA O MAC TOOLBOX

Observações:

- Neste anexo são descritas, usando o Canonicus, as classes concretas necessárias para implementar as classes genéricas da FIU Canônica sobre a Mac Toolbox.

- Os atributos definidos nas Classes Concretas que não possuam tipos correspondentes aos tipos previstos no Canonicus indicam declarações de variaveis necessárias para a execução das rotinas da Mac Toolbox e seus tipos são tipos relacionados a Mac Toolbox.

- O corpo das rotinas consiste basicamente de chamadas a funções da Mac Toolbox que implementam as rotinas deferidas das classes genéricas.

```
Concret Class Janela_Mac
Inherit Janela
Export

     Porta, Criar, Remover, Exibir, Esconder,
     Selecionar, Trocar_Pos, Realçar, Des_selecionar,
     Mover, Crescer, Inic_Atual, Fim_Atual

Feature

Tipo : INTEGER; -- constante inteira relativa a
                -- tipo pré-definido de janela
                -- (RDocProç etc ..)
Retângulo :  Coord;
          -- limites para movimentacao da janela
wRecord : Ptr;
badRect : Rect;
goodRect: Rect;
badRgn : Rgn;
goodRgn : Rgn;
Icone_assoc : BOOLEAN is FALSE;
Porta: GrafPort is GetWMgrPort(Porta);


Criar() is
do
  Current.Id:=NewWindows(@wRecord,Retângulo,Titulo,
                visisvel,tipo, Prox_jan, TRUE, 0 );
  SetPort(Porta);
end
```

```
Criar (recurso) is
do
  Id:=GetNewWindows(recurso, @wRecord, Prox_jan );
end

Remover () is
do
  DisposeWindow( Current.Id);
  |
  CloseWindow (Current.Id);
end

Exibir () is
do
  ShowWindow(Current.Id);
  |
  ShowHide (Current.Id, TRUE);
end

Esconder () is
do
  HideWindow(Current.Id);
  |
  ShowHide (Current.Id,FALSE);
end

Selecionar () is
do
  SelectWindow (Current.Id);
end

Trocar_Pos (Outra_jan) is
do
  SendBehind (Current.Id, Outra_jan);
  |
  BringToFront (Current.Id);
end

Realçar () is
do
  HiliteWindow ( Id, TRUE );
end

Des_selecionar () is
do
  HIliteWindow ( Id, FALSE );
end
```

```
Mover (coordx, coordy) is
local Coord : Point;
do
  Coord.x := coordx;
  Coord.y := coordy
  MoveWindow ( Id, Coord, FALSE );
end

Mover (pt_mousex, pt_mousey) is
local Pt_mouse : Point;
do
  GlobalToLocal(pt_mousex);
  GlobalToLocal(pt_mousey);
  Pt_mouse.x := pt_mousex;
  Pt_mouse.y := pt_mousey;
  DragWindow( Id , Pt_mouse, Retângulo  );
end

Crescer ( coordx, coordy) is
local Coord : Point;
do
  Coord.x := coordx;
  Coord.y := coordy
  SizeWindow ( Id , Coord, TRUE );
end

Crescer (pt_mousex, pt_mousey) is
local Pt_mouse : Point;
do
  GlobalToLocal(pt_mousex);
  GlobalToLocal(pt_mousey);
  Pt_mouse.x := pt_mousex;
  Pt_mouse.y := pt_mousey;
  GrowWindow ( Jan_Id, Pt_mouse, Retângulo );
end

Inic_Atual () is
do
  InvalRect(badRect) | InvalRgn(badRgn);
  ValRect(goodRect) | ValRgn(goodRgn);
  Begin_Update(Id)
end

Fim_Atual () is
do
  if Tipo = TIPO_COM_SIZE_BOX -- um tipo com size box
  then DrawGrowIcon(Id); DrawControls(Id);
  end
  EndUpdate(Id);
end
```

```
Concret Class Caixa_Diálogo_Mac
Inherit Caixa_DIálogo
Export

     Criar , Remover

Feature
dStorage : Ptr;
Ret_Caixa: Rect;
     -- Coordenadas do retângulo da Caixa de dialogo

Criar () is
do
  Id := NewDialog (dStorage, Ret_Caixa,titulo,
                   visivel, jan_dialogo.tipo,
                prox_dialogo, TRUE, 0, Lista_Itens );
end

Criar () is
do
  GetNewDialog ( recurso, dStorage, prox_dialogo );
end

Remover () is
do
  CloseDialog (Id);
  |
  DisposDialog (Id);
end



Concret Class Cx_Diálogo_Modal_Mac
Inherit Caixa_de_Diálogo_Mac
Export

     Criar

Feature

Criar () is
local temp : INTEGER;
     -- temporaria para retorno nao usado
do
  ModalDialog (NIL, temp );
end
```

```
Concret Class Alerta_Mac
Inherit Caixa_de_Diálogo_Mac
Export

     Alerta , Bip

Feature

Alerta (tipo)  is
do
  if tipo = STOP then
     Id:= StopAlert (recurso, NIL );
  elseif tipo = NOTE then
     Id:= NoteAlert (recurso, NIL );
  elseif tipo = CAUTION then
     Id := CautionAlert (recurso, NIL );
  else
     Id := Alert (recurso, NIL );
end

Bip () is
do
     -- automático no Mac para alertas
end;



Concret Class Menu_Mac
Inherit Menu
Export

     Remover

Feature
Retângulo : Rect;
     -- area da janela em que o menu eh exibido


Remover () is
do
  DisposeMenu (Id);
end
```

```
Concret Class Barra_Menu_Mac
Inherit Barra_Menu, Menu_Mac
Export

     Criar, Exibir, Selecionar, Des_Selecionar,
     Ins_Item, Rem_Item, Rem_todos_itens, Habil_Item,
     Desabil_Item
Feature

Criar () is
do
  Id:=GetNewMBar (recurso);
  SetMenuBar(Id);
end


Exibir () is
do
  DrawMenuBar;
end

Selecionar ()  is
do
  HiliteMenu(Id)
end

Des_Selecionar () is
do
  HiliteMenu(0)
end

Ins_Item (menu, pos) is
do
  InsertMenu(menu, pos)
end


Rem_Item (nitem) is
local menu : INTEGER;
do
  menu := Lista_itens (nitem);
  DeleteMenu(menu);
end

Rem_todos_itens () is
do
  ClearMenuBar;
end

Habil_item (num_item)  is
do
  EnableItem( Id, 0);
end
```

```
Desabil_item (num_item)  is
do
  DisableItem( Id, 0);
end

Concret Class Menu_Popup_Mac
Inherit Menu_Popup , Menu_Mac
Export

     Criar, Exibir, Ins_Item, Rem_Item,
     Habil_Item, Desabil_Item, Marcar_Item,
     Desmarcar_Item, Alter_Item, Obter_item

Feature

Marca : Char;  -- codigo ASCII da marca

Criar (texto) is
do
  Idx_menu := Idx_menu + 1;
  Id := NewMenu( Idx_menu, texto );
  titulo := texto ;
end

Criar () is
do
  Id:= GetMenu (recurso)
end

Exibir ()  is
do
  -- feito pelo MenuSelect no Manipulador de Eventos
end

Ins_Item (texto)  is
do
  AppendMenu(Id, NIL);
  SetItem ( Id, LAST, texto);
end


Ins_Item (tipo_recurso)  is
do
  AddResMenu(Id, tipo_recurso)
end


Ins_Item (texto, pos) is
do
  InsertResMenu(Id, NIL, pos);
  SetItem( Id, pos, texto );
end
```

```
Rem_Item (pos)  is
do
  SetItem (Id, pos , BRANCOS);
end

Habil_item (num_item)  is
do
  EnableItem (Id, num_item);
end

Desabil_item (num_item) is
do
  DisableItem (Id, num_item);
end

Marcar_Item (num_item) is
do
  CheckItem (Id, num_item, TRUE);
  |
  SetItemMark (Id, num_item, Marca );
end

Desmarcar_Item (num_item) is
do
  CheckItem (Id, num_item, FALSE);
  |
  SetItemMark (Id, num_item, 0 );
end

Alter_Item (num_item, novo_texto) is
do
  SetItem (Id,num_item, novo_texto );
end

Obtem_item (num_item) is
do
  GetItem(Id, num_item, nome );
  result := nome;
end


Concret Class Controle_Mac
Inherit Controle
Export

     Criar, Remover, Exibir, Realçar, Esconder,
     Mover, Crescer, Marcar, Desmarcar
```

```
Feature

ret_ctrl : Rect;
     -- pt superior esquerdo e pt inferior direito
intensidade : 1 .. 253;
     -- intensidade  do Realce (hilite)
Ret_limite, Slop_Rect : Rect;
     -- retangulos para movimentar controle

Criar () is
do
  Id := NewControl (jan_exib,ret_ctrl, titulo,
                visivel, valor, 0, 0, tipo , 0 );
end

Criar (recurso) is
do
  Id := GetNewControl ( recurso, jan_exib);
end

Remover () is
do
  DisposeControl (Id);
end

Exibir () is
do
  ShowControl (Id); -- torna visivel
  DrawControl (jan_exib);
          -- exibe os controles visiveis
end

Realçar () is
do
  HiliteControl (Id , Intensidade );
end

Esconder () is
do
  HideControl (Id);
end


Mover (coordx,coordy) is
local Coord : Point;
do
  Coord.x := coordx;
  Coord.y := coordy
  MoveControl ( Id, coord );
end
```

```
Mover (pt_mousex, pt_mousey) is
local Pt_mouse : Point;
do
  GlobalToLocal(pt_mousex);
  GlobalToLocal(pt_mousey);
  Pt_mouse.x := pt_mousex;
  Pt_mouse.y := pt_mousey;
  DragControl (Id, Pt_mouse, Ret_limite,
               Slop_Rect, NO_RESTR );
end

Crescer (coordx,coordy) is
local Coord : Point;
do
  Coord.x := coordx;
  Coord.y := coordy
  SizeControl (Id, Coord);
  ret_ctrl.inf_dir := Coord;
end

Crescer (pt_mousex, pt_mousey) is
local Pt_mouse: Point;
do
  GlobalToLocal(pt_mousex);
  GlobalToLocal(pt_mousey);
  Pt_mouse.x := pt_mousex;
  Pt_mouse.y := pt_mousey;
  SizeControl (Id,  Pt_mouse);
  ret_ctrl.inf_dir := Pt_mouse;
end

Marcar ( Id ) is
do
  SetCtlValue (Id, 1 );
end

Desmarcar ( Id ) is
do
  SetCtlValue (Id, 0);
end



Concret Class Texto_Mac
Inherit Texto
Export

     Fonte, Estilo, Tam, Tipo_alinhamento, Criar,
     Remover, Exibe_Cursor, Selecionar, Ativar,
     Desativar, Ins_txt, Alt_txt, Rem_txt, Cop_txt,
     Corta_txt, Cola_txt, Exibir, Alinha, Atualizar,
     Scroll, ParaClip, DeClip
```

```
Feature

Fonte : Font;
Estilo: Style;  -- (italico, bold, etc)
Tam : INTEGER;  -- constante com valor
                -- 8,12,16,18,20,24,36, ...
Ret_Visao, Ret_Desenho : Rect;
     -- View_rect e Destination_rect
Tipo_alinhamento : INTEGER;
     -- constante com valor TeJustLeft,
     -- TeJustCenter ou TeJustRight
Porta_txt : GrafPort;

Criar () is
do
  Id := TENew (Ret_Visao, Ret_Desenho );
end

Remover () is
do
  TEDispose (Id);
end

Exibe_Cursor () is
do
  TEIdle (Id);
end

Selecionar (pt_mousex, pt_mousey) is
local Pt_mouse : Point;
do
  GlobalToLocal(pt_mousex);
  GlobalToLocal(pt_mousey);
  Pt_mouse.x := pt_mousex;
  Pt_mouse.y := pt_mousey;
  TEClick (Pt_mouse, TRUE, Id );
end

Selecionar (  pos_inic , pos_fim ) is
do
  TESetSelect (pos_iniç pos_fim, Id);
end

Ativar () is
do
  TEActivate (Id);
end


Desativar () is
do
  TEDesactivate (Id);
end
```

```
Ins_txt (texto) is
local tam:INTEGER;
do
  tam := lenght (texto);
  TEInsert (texto,tam, txt_Id);
  InValRect(Ret_visao);
end

Alt_txt (texto_novo) is
do
  TEKey (texto_novo, Id);
  InValRect(Ret_visao);
end

Rem_txt () is
do
  TEDelete (Id);
  InValRect(Ret_visao);
end

Cop_txt () is
do
  TECopy (Id);
end

Corta_txt () is
do
  TECut (Id);
  InValRect(Ret_visao);
end

Cola_txt () is
do
  TEPaste(Id)
  InValRect(Ret_visao);
end

Exibir (texto,coordx, coordy) is
local tam:INTEGER;
      Coord : Point;
do
  tam := lenght (texto);
  Coord.x := coordx;
  Coord.y := coordy;
  TextBox ( texto, tam, Coord, tipo_alinhamento);
  InValRect(Ret_visao);
end
```

```
Exibir () is
do
  TextBox (Id,comprim_txt,ret_visao,tipo_alinhamento);
  InValRect(Ret_visao);
end

Alinha (alinhamento) is
do
  TESetJust (alinhamento, Id);
  InValRect(Ret_visao);
  tipo_alinhamento := alinhamento;
end


Atualizar () is
do
  EraseRect (Porta_txt);
  TEUpdate (Porta, Id);
end


Scroll (tipo) is
local Pag:INTEGER is <valor>;
     -- +/- PAG ou +/- LIN
linha:INTEGER is 1;
tam_scroll:INTEGER;
do
  if tipo < 0
  then tam_scroll := -linha
  else tam_scroll := linha; end
  tam_scroll := (y1.visao - y2.visao) * tam_scroll;
  TEScroll (0, tam_scroll, Id);
  InValRect(Ret_visao);
end


ParaClip ()  is
do
  TEToScrap();
end


DeClip ()  is
do
  TEFromScrap();
end
```

```
Concret Class Manip_Eventos_Mac
Inherit Manip_Eventos
Export

     Qual_Evento, evento

Feature

evento: EventRecord;
MaskChar: Char ;     -- Mascara para teclado
Timestamp: INTEGER;
janela: WindowPtr;

Qual_Evento(janela) is
local longa : LONGINT;
      flag : BOOLEAN;
      int : INTEGER;
      aux : CHAR;
do
  timestamp := timestamp + 1;
  If GetNextEvent (TODOS_EVENTOS, evento) then
     if evento.what = MouseDown then
        agente := USUARIO;
        int := FindWindow(evento.where, janela);
        if int = InSysWindow then
           SystemClick(evento,janela);
           -- tratamento de DA feito por SystemClick
        elsif int = InMenuBar then
           tipo := NULO;
           longa := MenuSelect(evento.where);
           item := LoWord(longa);
           menu := HiWord(longa);
           if menu /= 0 then
              tipo := SELECAO_MENU;
           end;  -- menu /= 0
        elsif int = InDrag then
           tipo := MOVER_JANELA;
           coord := evento.where;
        elsif int = InGrow then
           tipo := CRESCER_JANELA;
        elsif int = InGoAway then
           tipo := NULO;
           flag := TrackGoAway(janela,evento.where);
           if flag = TRUE then
              tipo := FECHA_JANELA;
           end;
```

```
    elsif int = InContent then
       tipo := NULO;
       if lista.itens /= NIL then
           -- ha controles na janela
          GlobalToLocal(evento.where);
          FindControl(evento.where,
                      janela,controle);
          if controle /= NIL then
             TrackCOntrol(controle,
                      evento.where,NIL);
             tipo := CLICK_CONTROLE;
             end;
          end; -- lista.itens
        if janela.ativa = TRUE and
           PtInRect(evento.where,
           texto.ret_visao) then
           GlobalToLocal(evento.where);
           flag := FALSE;
           if BitAnd(evento.modifiers,
              shiftkey) /= 0 then
              flag := TRUE;
                -- tecla "SHIFT" pressionada
              end;
           TEClick(evento.where,flag,texto.Id);
           tipo := SELECAO_TEXTO;
           end; -- janela ativa
        if janela.ativa = FALSE then
           tipo := SELECAO_JANELA;
        if IsDialogEvent(evento) then
           if DialogSelect(evento,
              caixa_dialogo,
              item) =  TRUE then
              tipo := CLICK_DIALOGO;
              end;  -- DialogSelect
           end;    -- IsDialogEvent
        if WaitMouseUp = TRUE  -- duplo click then
           tipo := tipo + CLICK_DUPLO;
           end;
        end; InContent
    end -- MouseDown
elsif evento.what = KeyDown or
    evento.what = AutoKey then
    agente := USUARIO;
    tipo := NULO;
    char := BitAnd(evento.message,MaskChar);
    aux := BitAnd(evento.modifiers, cmdkey);
    if aux /= 0 then  -- tecla de comando
       longa := MenuKey(char);
       item := LoWord(longa);
       menu := HiWord(longa);
       if menu /= 0 then
          tipo := SELECAO_MENU;
          end;
       end; -- aux
```

```
      if IsDialogEvent(evento) then
         if DialogSelect(evento,
            caixa_dialogo,item) then
            tipo:=ED_TEXTO_DLG;
            end;
         end; -- IsDialogSelect
      if janela.ativa = TRUE then
         tipo:=EDICAO_TEXTO;
         end;
      end;  -- KeyDown e AutoKey
   elsif evento.what = ActivateEvt then
      agente := APLICACAO;
      tipo := DESATIVAR_JANELA;
      if BitAnd(evento.modifiers,
         activeflag) /= 0 then
         tipo := ATIVAR_JANELA;
         end;
      end; -- ActivateEvt
   elsif evento.what = UpdateEvt then
      agente := APLICACAO;
      tipo := ATUALIZAR_JANELA;
      end; -- UpdateEvt
   elsif evento.what = DiskEvt then
      agente := APLICACAO;
      tipo := INS_DISCO;
      end; -- DiskEvt
   elsif evento.what = KeyUp then
      agente := USUARIO;
      tipo := TECLA_SOLTA;
      end; -- KeyUp
   elsif evento.what = MouseUp then
      agente := USUARIO;
      tipo := MOUSE_SOLTO;
      end; -- MouseUp
   elsif evento.what = DriverEvt then
      agente := APLICACAO;
      tipo := EVENTO_DISPOSITIVO;
      end; -- DriverEvt
   elsif evento.what = AppleEvt then
      agente := APLICACAO;
      tipo := EVENTO_APLICACAO;
      end; -- AppleEvt
   end; -- GetNextEvent
Result := tipo;
end; -- Qual_Evento
```

ANEXO E CLASSES CONCRETAS PARA O MS WINDOWS

Observações:

- Neste anexo são descritas, usando o Canonicus, as classes concretas necessárias para implementar as classes genéricas da FIU Canônica sobre o MS Windows.

- Os atributos definidos nas Classes Concretas que não possuam tipos correspondentes aos tipos previstos no Canonicus indicam declarações de variáveis necessárias para a execução das rotinas do MS Windows e seus tipos são tipos relacionados aos tipos do MS Windows.

- O corpo das rotinas consiste basicamente de chamadas a funções do MS Windows que implementam as rotinas deferidas das classes genéricas.

```
Concret Class Janela_MSW
Inherit Janela
Export

     Cor_fundo, lpfnWndProc, Criar, Remover,
     Exibir, Esconder, Selecionar, Des_selecionar,
     Mover, Crescer, Troca_pos, Inic_Atual,
     Fim_Atual, Realçar

Feature

Coordx : INTEGER;
     -- X do ponto superior esquerdo do retangulo
Coordy : INTEGER;
     -- Y do ponto superior esquerdo do retangulo
Largura : INTEGER;  -- largura do retangulo
Altura : INTEGER;   -- altura do retangulo
Cor_fundo : HBRUSH; -- cor de fundo da janela
tipo : WORD;        -- "style" em /JAM 87/
Menu_jan : Menu_MSW;
badRect : Rect;
badRgn : Rgn;
goodRect : Rect;
goodRgn : Rgn;
Icone_assoc : BOOLEAN is IsIconic(Id);
Area : LPPAINTSTRUCT;    -- usado para atualizacao
Modulo : Ptr is GetModuleHandle (<nome_da_aplicacao>);
     -- hInstance em /JAM 87/
Visivel: BOOLEAN is IsWindowVisible(Id);
lpParam: LPMSG;
lpfnWndProc : FARPROC;
     -- apontador para rotina que trata
     -- mensagens da janela
```

```
Criar ()  is
do
  Id := CreateWindow ( Classe, titulo, tipo, Coordx,
                       Coordy, largura, altura, Jan_pai,
                       Menu_jan, Modulo, lpParam );
end

Remover () is
do
  DestroyWindow  (Id);
end

Exibir () is
do
  If Icone_assoc = TRUE
  then OpenIcon (id)
  end
  |
  ShowWindow(Id, SHOW_ICONWINDOW);
end

Exibir () is
do
  ShowWindow (Id, SHOW_FULLSCREEN);
end

Esconder () is
do
  If Icone_assoc = TRUE then
     CloseWindow (Id)
     |
     ShowWindow(Id,HIDE_WINDOW);
  endif
  visivel := FALSE;
end

Selecionar () is
do
  SetActiveWindow (Id)
  |
  EnableWindow(Id,TRUE);
end

Des_selecionar () is
do
  EnableWindow(Id,FALSE);
end

Mover (coordx, coordy) is
do
  MoveWindow (Jan_Id, coordx, coordy, largura,
              altura, TRUE );
end
```

```
Mover (pt_mousex, pt_mousey) is
local Pt_mouse : Point;
do
  Pt_mouse := GetCUrsorPos(pt_mouse);
  MoveWindow (Jan_Id, Pt_mouse.x, Pt_mouse.y,
               largura, altura, TRUE );
end

Crescer (coordx, coordy) is
local  nova_largura, nova_altura : INTEGER;
do
  nova_altura := external Calc_altura (altura, coordy)
                 languagem < L > name "calc_alt";
  nova_largura:= external Calc_largura(largura,coordx)
                 languagem < L > name "calc_larg";
  MoveWindow (Jan_Id, coordx, coordy,
              nova_largura, nova_altura, TRUE );
end


Crescer (pt_mousex, pt_mousey)  is
local  nova_pos, nova_largura, nova_altura : INTEGER;
       Pt_mouse : Point;
do
  Pt_mouse := GetCursorPos (pt_mouse);
  nova_altura := external Calc_altura(altura,Pt_mouse)
                 languagem < L > name "calc_alt";
  nova_largura:= external  Calc_largura ( largura,
                                    Pt_mouse)
                 languagem < L > name "calc_larg";
  MoveWindow (Jan_Id, nova_pos.coordx,nova_pos.coordy,
               nova_largura, nova_altura, TRUE );
end


Troca_pos () is
do
  BringWindowToTop(Id);
end

Inic_Atual () is
do
  ValidateRect(Id,goodRect) | ValidateRgn(Id,goodRgn);
  InvalidateRect(Id,badRect) |
  InvalidateRgn(Id,badRgn);
  BeginPaint(Id, Area );
end

Fim_Atual () is
do
  UpdateWindow (Id);
  EndPaint(id,Area);
end
```

```
Realçar () is
do
  FlashWindow (Id, TRUE);
end


Concret Class Caixa_Diálogo_MSW
Inherit Caixa_Diálogo
Export

    classe_alerta, Criar, Remover

Feature
jan_dialogo : Janela_MSW;
classe_alerta : INTEGER;
     -- constante com valor
     -- MB_OK, MB_OKCANCEL, MB_RETRYCANCEL,
     -- MB_YESNO, MB_ABORTRETRYIGNORE,
     -- MB_YESNOCANCEL, MB_ICONHAND,
     -- MB_DEFBUTTON1, MB_APPLMODAL,
     -- MB_DEFBUTTON2, MBSYSTEMMODAL,
     -- MB_DEFBUTTON3.

modelo : LPSTR is "<nome_da_aplicacao>+"_DLG";
funcao : PROC;

Criar () is
local modulo: Ptr;
do
  jan_dialogo := janela.Criar();
  modulo := jan_dialogo.modulo;
  funcao := MakeProcInstance
          ((FARPROC) Dlg<nome_aplicacao>Box,módulo);
  Id:= CreateDialog (jan_dialogo.modulo, modelo,
                     jan_dialogo, funcao );
end

Remover () is
do
  EndDialog (Id, TRUE);
end


Concret Class Cx_Diálogo_Modal_MSW
Inherit Cx_Diálogo_Modal, Caixa_diálogo_MSW
Export

     Criar
```

```
Feature

Criar () is
do
  Id := DialogBox (jan_dialogo.modulo, modelo,
                   jan_dialogo, funcao );
end



Concret Class Alerta_MSW
Inherit Alerta, Caixa_Diálogo_MSW
Export

     txt_alerta, Alerta, Bip

Feature
txt_alerta : STRING;

Alerta (tipo_alerta) is
do
  if tipo_alerta = NOTE then
     MessageBox(jan_dialogo,txt_alerta,
                modelo,MB_ICONASTERISK);
  elseif tipo_alerta = STOP then
     MessageBox(jan_dialogo,txt_alerta,
                modelo,MB_ICONEXCLAMATION);
  elseif tipo_alerta = CAUTION then
     MessageBox(jan_dialogo,txt_alerta,
                modelo,MB_ICONQUESTION);
  else
     MessageBox(jan_dialogo, txt_alerta,
                modelo, classe_alerta);
end

Bip () is
do
  MessageBeep (classe_alerta);
end



Concret Class Menu_MSW
Inherit Menu
Export

     Remover

Feature

Remover () is
do
  DestroyMenu (Id);
end
```

```
Concret Class Barra_Menu_MSW
Inherit Barra_Menu, Menu_MSW
Export

     Criar, Exibir, Selecionar, Des_selecionar,
     Ins_item, Habil_item, Desabil_item,
     Rem_item, Rem_todos_itens

Feature

Criar () is
do
  Id := GetMenu(jan_menu);
  SetMenu (jan_menu, Id);
  jan_menu.menu_jan := Id;
end

Exibir () is
do
  DrawMenuBar (jan_menu);
end

Selecionar (num_item) is
do
  HiliteMenuItem (jan_menu, Id, num_item,
              [MF_HILITE,MF_BYPOSITION]);
end

Des_Selecionar (num_item) is
do
  HiliteMenuItem (jan_menu, Id, num_item,
              [MF_UNHILITE,MF_BYPOSITION]);
end

Ins_Item (menu, nitem) is
do
  ChangeMenu( Id, nitem, menu, 0,
            [MF_APPEND,MF_BYPOSITION]);
end;

Rem_Item (num_item) is
do
  ChangeMenu (Id, num_item, NIL,0,
            [MF_DELETE,MF_BYPOSITION]);
end

Rem_todos_itens () is
do
  ChangeMenu (Id, ALL , NIL,0,
            [MF_DELETE,MF_BYPOSITION]);
end
```

```
Habil_item (nitem) is
do
  ChangeMenu( Id, nitem, NIL, 0,
            [MF_ENABLED,MF_BYPOSITION]);
end

Desabil_item (nitem) is
do
  ChangeMenu( Id, nitem, NIL, 0,
            [MF_DISABLED,MF_BYPOSITION]);
end


Concret Class Menu_Popup_MSW
Inherit Menu_PopUp, Menu_MSW
Export

     Criar, Exibir, Rem_Item,Ins_Item,
     Alter_Item, Habil_Item, Desabil_Item,
     Marcar_Item, Desmarcar_Item, Obter_Item

Feature

Criar (texto) is
do
  ChangeMenu( Id, num_item, texto, 0,
            [MF_STRING,MFBYPOSITION]);
end

Criar ()
do
  ChangeMenu(Id, num_item, NIL, 0,
            [MF_POPUP, MF_BYPOSITION]);
end

Exibir (num_menu) is
do
  GetSubMenu (Id, num_menu);
end

Rem_Item (num_item) is
do
  ChangeMenu (Id, num_item, NIL,0,
            [MF_DELETE,MF_BYPOSITION]);
end

Ins_Item (texto) is
do
  ChangeMenu (Id, 0, texto,0, MF_APPEND);
end
```

```
Ins_Item(texto,pos) is
do
  ChangeMenu( Id,0,texto,num_item,
            [MF_INSERT,MF_BYPOSITION]);
end

Ins_Item(tipo_rec) is
local item : INTEGER;
do
  item:=FindResource(jan_menu.modulo,
                     NIL, tipo_rec);
  ChangeMenu(Id,0,item,0,MF_APPEND);
end

Alter_Item (num_item,texto_novo) is
do
  ChangeMenu (Id, num_item, texto_novo,0, MF_CHANGE);
end

Habil_Item (num_item) is
do
  EnableMenuItem (Id,num_item,
                  [MF_ENABLED,MF_BYPOSITION]);
end


Desabil_Item (num_item) is
do
  EnableMenuItem (Id, num_item,
                  [MF_DISABLED,MF_BYPOSITION]);
end

Marcar_Item (num_item) is
do
  CheckMenuItem (Id, num_item,
                 [MF_CHECKED,MF_BYPOSITION]);
end

Desmarcar_Item (num_item) is
do
  CheckMenuItem (Id, num_item,
                 [MF_UNCHECKED,MF_BYPOSITION]);
end

Obter_Item (pos) is
local Tammax: INTEGER is 255;
do
  GetMenuString( Id, pos, nome, Tammax,
                MF_BYPOSITION );
  result := nome;
end
```

```
Concret Class Controle_MSW
Inherit Controle
Export

     Criar, Exibir, Realçar, Esconder,
     Marcar, Desmarcar, Crescer, Mover,
     Remover

Feature
Coordx : INTEGER;
     -- coord x do pt superior esquerdo do controle
Coordy : INTEGER;
     -- coord y do pt superior esquerdo do controle
largura : INTEGER;  -- largura do controle
altura : INTEGER;   -- altura do controle


Criar () is
do
  Id:=SendDlgItemMessage(WM_GETDLGCODE)
end


Exibir () is
do
     -- automatico junto com a caixa de
     -- dialogo que o contém
end

Realçar () is
do
  SendMessage(jan_exib, BM_SETSTATE,HILITE,0);
end

Esconder() is
do
  SendMessage(jan_exib, BM_SETSTATE,UNHILITE,0);
end

Marcar () is
do
  SendMessage(jan_exib, BM_SETCHECK,CHECKMARK,0);
  |
  CheckDlgButton (jan_exib, Id, 0);
end

Desmarcar () is
do
  SendMessage(jan_exib, BM_SETCHECK,UNCHECKMARK,0);
  |
  CheckDlgButton (jan_exib, Id, 1);
end
```

```
Crescer () is
do
     -- junto com a janela a que pertence
end

Mover () is
do
     -- junto com a janela a que pertence
end

Remover () is
do
     -- junto com a janela a que pertence
end
```

Muitas mensagens de controle são enviadas da aplicação à janela de diálogo que o contém por meio da função **SendMessage.**

```
Concret Class Texto_MSW
Inherit Texto
Export

     Criar, Remover, Exibe_Cursor, Selecionar,
     Ativar, Desativar, Ins_txt, Alt_txt,
     Rem_txt, Cop_txt, Corta_txt, Cola_txt,
     Alinha, Atualizar, Scroll, Exibir,
     ParaClip, DeClip

Feature
ret_edicao : Rect;
Jan_edicao : Janela;
Formato_cursor: Bitmap;
Largura_cursor : INTEGER;
Altura_cursor : INTEGER;
hDC : HDC;
     -- referencia ao contexto de dispositivo desejado
Formato_clipboard : WORD;

Criar (coordx, coordy) is
do
  SetFocus (jan_edicao);
  CreateCaret  (jan_edicao, formato_cursor,
                largura_cursor, altura_cursor);
  SetCaretPos(coordx, coordy);
  ShowCaret(jan_edicao);
end

Remover () is
do
  DestroyCaret();
end
```

```
Exibe_Cursor () is
do
  ShowCaret (jan_edicao);
end

Selecionar (coordx, coordy) is
do
  ret_edicao.x1 := coordx;
  ret_edicao.y1 := coordy;
  ret_edicao := SendMessage(jan_edicao,
                            EM_GETSEL, 0, 0)
  SendMessage(jan_edicao, EM_SETSEL, 0,
              ret_edicao);
end

Ativar () is
do
  -- automatico
end

Desativar () is
do
     -- automatico
end

Ins_txt (texto) is
do
  external insere_texto (texto)
  language < L > name "insere_txt";
     -- a ser feita por um usuário projetista
end

Alt_txt (texto_novo) is
do
  SendMessage(jan_edicao, EM_REPLACESEL, 0,
              textonovo);
end

Rem_txt () is
do
  SendMessage(jan_edicao, WM_CLEAR, 0, 0);
end

Cop_txt () is
do
  SendMessage(jan_edicao, WM_COPY, 0, 0);
end

Corta_txt () is
do
  SendMessage(jan_edicao, WM_CUT, 0, 0);
end
```

```
Cola_txt () is
do
  SendMessage(jan_edicao, WM_PASTE,0, 0)
end

Alinha (alinhamento) is
do
  DrawText( hDC, comprim_txt, ret_edicao,
          alinhamento );
end

Atualizar () is
do
  UpdateWindow (jan_edicao);
end

Scroll (tipo_scroll) is
local flag_scroll : INTEGER is 1;
do
  if tipo_scroll < 0 then
     flag_scroll := (-1);
     tipo_scroll := tipo_scroll * (-1);
  endif
  if tipo_scroll = PAG then
     flag_scroll := SB_PAGE * flag_scroll
          -- SB_PAGEUP e SB_PAGEDOWN
  endif
  if tipo_scroll = LIN then
     flag_scroll := SB_LINE * flag_scroll
          -- SB_LINEUP e SB_LINEDOWN
  endif
  SendMessage(jan_edicao,EM_SCROLL, flag_scroll,0);
end

Exibir () is
do
  TextOut(hDC,ret_edicao.x1,ret_edicao.y1,
          texto, comprim_txt);
end

Exibir (texto, coord) is
local tam: INTEGER;
do
  tam := lenght(texto);
  TextOut (hDC, coord.x1, coord.y1, texto, tam);
end


ParaClip () is
do
  OpenClipBoard (jan_edicao);
  SetClipBoardData(Formato_clipboard, Id);
end
```

```
DeClip () is
do
  GetClipBoardData(Formato_clipboard);
  CloseClipBoard ();
end
```

Muitas mensagens de texto são enviadas da aplicação à janela de controle de edição por meio da função SendMessage. O controle executa a tarefa requerida e devolve um valor à rotina que fez a chamada.

```
Concret Class Manip_Evento_MSW
Inherit Manip_Evento
Export

     Qual_Evento

Feature

mensagem : MSG;
janela : INTEGER;
filtro_min,     -- valor minimo de mensagem examinada
filtro_max : INTEGER -- valor maximo
posicao : Point;
     -- posicao do mouse em coordenadas x e y

Criar (jan) is
do
  janela := jan;
     -- Manipulador de Eventos associado a uma janela
end

Qual_Evento() is
local caracter: CHAR;
      msg : INTEGER;
do
  timestamp := timestamp + 1;
  GetMessage(mensagem,janela,filtro_min,filtro_max);
  msg := mensagem. message;  -- codigo de mensagem
  if msg = WM_Active or
     msg = WM_Activateapp then
     agente := APLICACAO;
     tipo := DESATIVAR_JANELA;
     if wParam /= 0 then
        tipo := ATIVAR_JANELA;
        end;
     janela := LoWord(lParam)
     end; -- WM_Active
```

```
elsif msg = WM_Close then
   agente := APLICACAO;
   tipo := FECHAR_JANELA;
   end;
elsif msg = WM_Move then
   agente := APLICACAO;
   tipo := MOVER_JANELA;
   end;
elsif msg = WM_Paint then
   agente := APLICACAO;
   tipo := ATUALIZAR_JANELA;
   end;
elsif msg = WM_Size then
   agente := APLICACAO;
   tipo := CRESCER_JANELA;
   end;
elsif msg = WM_Char then
   agente := USUARIO;
   caracter := HiWord(mensagem.lParam);
   tipo := TECLA_SOLTA;
   if TestBits(caracter,16) = 0  then -- bit 32
      tipo:= TECLA;
      end;
   char := mensagem.wParam;  -- tecla digitada
   end;
elsif msg = WM_Command then
   agente := USUARIO;
   caracter := HiWord(mensagem.lParam);
   if caracter = 0 or
      caracter = 1 then
      tipo := SELECAO_MENU;
      item := mensagem.wParam;
      menu := LoWord(mensagem.lParam);
      end;
   else
      tipo := CLICK_CONTROLE;
      controle := mensagem.wParam;
   end;  -- caracter = 0
elsif msg = WM_HScroll then
   agente := USUARIO;
   posicao := LoWord(lParam);
   tipo := SCROLL_HORIZ;
   end;
elsif msg = WM_Scroll then
   agente := USUARIO;
   posicao := LoWord(lParam);
   tipo := SCROLL_VERT;
   end;
elsif msg = WM_LButton then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := CLICK_MOUSE + BOTAO_1;
   end;
```

```
elsif msg = WM_MButton then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := CLICK_MOUSE + BOTAO_2;
   end;
elsif msg = WM_RButton then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := CLICK_MOUSE + BOTAO_3;
   end;
elsif msg = WM_ButtonDblCkl then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := CLICK_MOUSE + DUPLO_CLICK+ BOTAO_1;
   end;
elsif msg = WM_MButtonDblCkl then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := CLICK_MOUSE + DUPLO_CLICK+ BOTAO_2;
   end;
elsif msg = WM_RButtonDblCkl then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := CLICK_MOUSE + DUPLO_CLICK+ BOTAO_3;
   end;
elsif msg = WM_LButtonUp then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := MOUSE_SOLTO + BOTAO_1;
   end;
elsif msg = WM_MButtonUp then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := MOUSE_SOLTO + BOTAO_2;
   end;
elsif msg = WM_RButtonUp then
   agente := USUARIO;
   posicao.x := LoWord(lParam);
   posicao.y := HiWord(lParam);
   tipo := MOUSE_SOLTO + BOTAO_3;
   end;
```

```
elsif msg = WM_ShowWindow then
   agente := USUARIO;
   tipo := SELECAO_JANELA;
   end;
else
   -- Se  nao eh nenhuma anterior
   -- deixa o MS Windows processar a
         mensagem
Result := tipo;
end; -- Qual_Evento
```

ANEXO F GRAMÁTICA DA NOTAÇÃO DO CANONICUS

Obs: As produções em que a Gramática do Canonicus é diferente da de Eiffel estão marcadas pelo símbolo $.

```
Class Declaration   = Class_header
                      [Exports]
                      [Parents]
                      [Features]
                      end

Class_header        = [Level] Class Class_name
$Level              = Generic | Concret
Class_name          = Identifier

Exports             = export Export_list
Export_list         = { Export_item "," ...}
Export_Item         = Feature_name
Feature_name        = Identifier

Parents             = inherit Parent_list
Parent_list         = { Parent ";" ...}
Parent              = Class_type
Class_type          = Class_name

Features            = feature {Feature_declaration";"}
Feature_declaration = Feature_name
                      [ Formal_arguments ]
                      [ Type_mark ]
                      [ Feature_value_mark ]

Formal_arguments    = Entity_decl_list
Entity_decl_list    = {Entity_decl_group";"...}
Entity_decl_group   = { Identifier "," ...} Type_mark
Type_mark           = ":" Type
$Type               = INTEGER | BOOLEAN | CHAR
                    | REAL | STRING
                    | Class_Type |
                    | [ FIU_Type ]

Class_Type          = Class_name
$FIU_Type           = <Specific FIU Type>
    /* definido somente para classes concretas */

Feature_value_mark  = is Feature_value
Feature_value       = Constant | Routine
```

```
Constant              = Integer_constant |
                        Character_constant |
                        Boolean_constant |
                        Real_constant |
                        String_constant

Integer_constant      = [Sign] Integer
Sign                  = "+" | "-"
Character_constant    = "'" Character "'"
Boolean_constant      = TRUE | FALSE
Real_constant         = [Sign] Real
String_constant       = '"' String '"'

$Routine              = [Local_variables]
                        [Body]
                        end

Local_variables       = Entity_decl_list
$Body                 = do Compound
                        [ { "|" Compound ... } ]
Compound              = { Instruction ";" ...}

$Instruction          = Call |
                        Assignment |
                        Conditional

Call                  = Qualified_call |
                        Unqualified_call
Qualified_call        = Expression "." Unqualified_call
$Unqualified_call     = Feature_name [Actuals] |
                        Externals
Actuals               = "(" Expression_list ")"
Expression_list       = {Expression Separator ...}
Separator             = "," | ";"

Externals             = external External_list
External_list         = {External_decl ";" ... }
External_decl         = Feature_name
                        [ Formal_arguments ]
                        [ Type_mark ]
                        [ External_name ]
                        Language
External_name         = name String_constant
Language              = language String_constant

Assignment            = Entity ":=" Expression
Entity                = Identifier | Result

Expression            = {Unqualified_expr "."...}
Unqualified_expr      = Constant |
                        Entity |
                        Unqualified_call |
                        Current |
                        Operator_expr
```

```
Operator_expr       = Unary_expr |
                      Binary_expr |
                      Multiary_expr |
                      Parenthesized
Unary_expr          = Unary Expression
Unary               = not | "+" | "-"
Binary_expr         = Expression Binary Expression
Binary              = ^ | = | /= | < | > | <= | >=
Multiary_expr       = {Expression Multiary ...}

Multiary            = "+" | "-" | "*" | "/" |
                      and | or
Parenthesized       = "(" Expression ")"

Conditional         = if Then_part_list
                      [Else_part] end
Then_part_list      = { Then_part elsif ...}
Then_part           = Boolean_expression then Compound
Else_part           = else Compound
```

BIBLIOGRAFIA

/ANS 82/ ANSON, E. The Device Model of Interaction. Computer Graphics, New York, v. 16, n.3, p. 107-114, July 1982.

/APP 85/ APPLE Computer. Inside Macintosh V.I,II,III. Reading: Addison-Wesley, 1985.

/BAG 89/ BAGGIO,A. Uma Interface de gerenciamento para o Projeto Tranca. Porto Alegre: CIC-UFRGS, 1989. (Trabalho de Diplomação).

/BAR 86/ BARTH,P. An Object-Oriented Approach to Graphical Interfaces. ACM Transactions On Graphics, New York, v. 5, n. 2, p. 142-72, Apr. 1986.

/BEN 86/ BENNET, J.L. Tools For Building Advanced User Interface. ,IBM Systems Journal, Armonk, v. 25, n. 3/4, p. 354-68, July 1986.

/BÖS 87/ BÖSSER, T. Learning in Man-Computer Interaction - a review of the literature. Berlin: Springer-Verlag, 1987.

/BUX 83/ BUXTON,W.A. et al. Towards a Comprehensive User Interface Management System. Computer Graphics, New York, v. 17, n. 3, p. 35-42, July 1983.

/CAR 83/ CARROLL, J.M. Presentation and Form in User Interface Architecture. BYTE, Peterborough, v.8, n. 12, p. 113-22, Dec. 1983.

/CAR 85/ CARDELLI, L. ; PIKE, R. Squeak: a Language for Communicating with mice. ACM Computer Graphics, New York, v. 19, n. 3, p.199-204, July 1985. SIGGRAPH'85.

/CGR 87/ COMPUTER Graphics, New York, v. 21, n. 2, Apr. 1987.

/CRD 83/ CARD, S. et al. The Psichology of Human-Computer Interaction. HillsDale: Lawrence Erlbaum, 1983.

/CHEN 76/ CHEN,P.P.S. The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems, v.1, n.1, p. 9-36, Mar. 76.

/COH 86/ COHEN, B.; HARWOOD, W.T.; JACKSON, M.I. The Specification of Complex Systems. Reading: Addison-Wesley, 1986.

/COR 79/ COMPUTER Graphics, New York, v.13, n.3, Aug. 1979. Special Issue on CORE.

/COU 85/ COUTAZ, J. Abstractions for User Interface Design. Computer, New York, v. 18, n. 9, p. 21-34, Sept. 1985.

/COX 84/ COX, B. Message/Object Programming: an Evolutionary Change in Programming Technology. IEEE Software, Los Alamitos, v. 1, n. 1, p. 50-62, Jan. 1984.

/DAN 87/ DANCE, J.R. The Run-Time Structure of UIMS-Supported Applications. Computer Graphics, New York, v. 21, n. 2, p. 97-101, Apr. 1987.

/DAN 88/ DANFORTH, M. ; TOMLINSON, C. Type Theories and Object-Oriented Programming. Computing Surveys, New York, v. 20, n. 1, p. 69-96, Mar. 1988.

/DEH 81/ DEHINNG, Waltraud; ESSIG, Heidrun; MAAS, Susanne. The Adaptation of Virtual Man-Computer Interfaces to User Requirements in Dialogs. Berlin: Springer-Verlag, 1981. 142 p. Lecture Notes in Computer Science 110

/ESP 89/ ESPERANÇA, L. G. ; Favero, E.L.; Price, R.T. Um Algoritmo de Reconhecimento Incremental. Porto Alegre: CPGCC-UFRGS, 1989. Relatório de Pesquisa 117.

/FIS 88/ FISCHER, A. User Interface Design Methodologies. Using Software Development Tools. New Yor: John Wiley, 1988.

/FOL 87/ FOLEY, J. Transformations on a Formal Specification of User-Computer Interfaces. Computer Graphics, New York, v. 21, n. 2, p. 109-13, Apr. 1987.

/FPP 90/ FRAINER, A.S.; PIMENTA, M.S.; PRICE,R.T. Como Obter Portabilidade de Programas Interativos. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTACAO, 10., 1990, Vitória. Anais... Vitória: SBC, 1990. 611 p. p. 496-512.

/FRA 89/ FRAINER, A. S. Um Processador de Linguagens. Porto Alegre: CIC-UFRGS, 1989. (Trabalho de Diplomação).

/GIR 90/ GIRARDI, M. ; PRICE, R.T. O Paradigma de Desenvolvimento por Objetos. Revista de Informática Teórica e Aplicada, Porto Alegre, v. 1, n. 2, p. 69-98, Maio 1990.

/GKS 84/ COMPUTER Graphics, New York, v.18, Feb. 1984. Special Issue on GKS.

/GOL 83a/ GOLDBERG,A. Smalltalk 80: the Interactive Programming Environment. Reading: Addison-Wesley, 1983.

/GOL 83b/ GOLDBERG,A. Smalltalk 80: Creating a User Interface and Graphical Applications. Reading: Addison-Wesley, 1983.

/GOL 83c/ GOLDBERG,A. Smalltalk 80: the Language and Its Implementation. Reading: Addison-Wesley, 1983.

/GRE 84/ GREGG, W. The Apple Macintosh Computer. BYTE, Peterborough, v. 9, n. 2, p. 30-54, Feb. 84.

/GRE 86/ GREEN, M. A Survey of Three Dialogue Models. ACM Transactions on Graphics, New York, v. 5, n. 3, p. 244-75, July 86.

/GRE 87/ GREEN, M. Directions for User Interface Management Systems Research. Computer Graphics, New York, v. 21, n. 2, p. 113-16, Apr. 1987.

/HAH 89/ HARTSON, H.R.; HIX, D. Human-Computer Interface Development: Concepts and Systems for Its Management. Computing Surveys, New York, v. 21, n. 1, p. 5-92, Mar. 1989.

/HAR 89/ HARTSON, R. User Interface Control and Communication. IEEE Software, Los Alamitos, v. 6, n. 1, p. 62-70, Jan. 1989.

/HIL 87/ HILL,R.D. Some Important Features and Issues in User Interface Management Systems. Computer Graphics, New York, v. 21, n. 2, p. 116-20, Apr. 1987.

/HUD 86/ HUDSON, S. ; KING, R. A Generator of Direct Manipulation Office Systems. ACM Transactions on Office Information Systems, New York, v. 24, n. 2, p. 132-63, Apr. 1986.

/HUD 87/ HUDSON, S.E. UIMS Support for Direct Manipulation Interfaces. Computer Graphics, New York, v. 21, n. 2, p. 120-24, Apr. 1987.

/JAM 87/ JAMSA, K. Windows Programming Secrets. Berkeley: Osborne-McGraw-Hill, 1987.

/JOH 75/ Johnson, S.C. YACC - Yet Another Compiler-Compiler. Murray Hill: AT&T Bell Laboratories, 1975. Computing Science Technical Report 32.

/KOI 88/ KOIVUNEN, M.R. et al. HutWindows: an Improved Architecture For a User Interface Management System. IEEE Computer Graphics & Applications, Los Alamos, v. 8, n.1, p. 43-52, Jan. 1988.

/LIE 86/ LIEBERMANN, H. Using Prototypical Objects to Implement Shared Behaviour in Object Oriented Systems. Sigplan Notices, New York, v. 21, n. 11, p. 214-23, Nov. 1986.

/LIN 89/ LINTON, M. et al. Composing User Interfaces with Interviews. IEEE Computer, New York, v. 22, n. 2, p. 8-22, Feb. 1989.

/MAR 73/ MARTIN, J. Design of Man-Computer Dialogues, Englewood Cliffs: Prentice-Hall, 1973.

/MAR 90/ MARCUS,A. Designing Graphical User Interfaces. UNIXWORLD, p. 107-15, Aug. 90.

/MEY 86/ MEYROWITZ, N. Intermedia: the Architecture and Construction of an Object-Oriented Hypermedia System and Application Framework. Sigplan Notices, New York, v. 21, n. 11, p 186-201, Nov. 1986.

/MEY 88/ MEYER, B. Object-Oriented Software Construction, Englewood Cliffs: Prentice-Hall, 1988. Series in Computer Science.

/MOR 81/ MORAN, T.P. The Command Language Grammar: a Representation for The User Interface of Interactive Computer Systems. International Journal of Man-Machine Studies, v. 15, p. 3-50, 1981.

/MYE 86/ MYERS, B.A ; BUXTON,W.A. Creating Highly Interactive and Graphical User Interfaces by Demonstration. Computer Graphics, New York, v. 20, n. 4, p. 249-58, Aug. 1986.

/MYE 88/ MYERS,B. A Taxonomy of Window Managers User Interfaces. IEEE Computer Graphics & Application, Los Alamos, v. 8, n. 5, p. 65-84, Sept. 1988.

/MYE 89/ MYERS, B. User Interface Tools: Introduction and Survey. IEEE Software, Los Alamitos, v. 5, n. 1, p. 15-23, Jan. 1989.

/OLS 83/ OLSEN,D.R. ; DEMPSEY,E.P. SINGRAPH: a Graphical User Interface Generator. Computer Graphics, New York, v. 17, n. 3, p. 43-50, July 1983.

/OLS 84/ OLSEN, D. et al. A Context for User Interface Management. IEEE Computer Graphics & Applications, Los Alamos, v. 4, n. 2, p. 33-42, Dec. 1984.

/OLS 86/ OLSEN,D. MIKE: the Menu Interaction Kontrol Environment. ACM Transactions on Graphics, New York, v. 5, n. 4, p. 318-44, Oct. 1986.

/PET 89/ PETZOLD, C. et al. GUIs For DOS and OS/2. PC Magazine, New York, v. 8, n. 15, p. 112-131, Sept. 1989.

/PIM 90/ PIMENTA,M.S.; MELGAREJO,L.F.B. Uso de Um Sistema Hipertexto/Hipermídia como Modelo de Interface com o Usuário. In: JORNADAS DE INGENIERIA DE SISTEMAS INFORMATICOS Y DE COMPUTACION, 1., 1990, Quito, Equador. Anales... Quito, Equador, 1990, p. 188-93.

/PRI 87/ PRICE, R.T.; Favero, E.L. Uma Introdução a Linguagem de Especificação (LDE) do Ambiente de Desenvolvimento de Software. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÀO, 7., 1987, Salvador. Anais... Salvador: SBC/UFBA, 1987, 613 p. p. 538-49.

/PRI 88/ PRICE,R.T. Interface com o Usuário para o Projeto AMADEUS, Comunicacao Pessoal, jun 88.(não publicado).

/PUG 86/ PUGLIA, V. et al. Operating in a New Environment. PC Magazine, New York, v. 5, n. 4, p.108-15, Feb. 1986.

/RHY 87/ RHYNE,J. et al. Tools and Methodology for User Interface Development. Computer Graphics, New York, v. 21, n. 2, p. 137-42, Apr. 1987.

/SCG 86/ SCHEIFLER, R.W. ; GETTYS,J. The X Window System. ACM Transactions on Graphics, New York, v. 5, n. 2, p. 79-109, Apr. 1986.

/SCH 86/ SCHMUKER, K.J. MacApp: an Application Framework. BYTE, Peterborough, v. 11, n. 8, p. 189-193, Aug. 1986.

/SCH 91/ SCHUBERT, E.G. Uma Linguagem de Definição e Manipulação de Interfaces com o Usuário, Porto Alegre: CPGCC da UFRGS, 1991. (Dissertação de Mestrado).

/SHN 87/ SHNEIDERMAN, B. Designing the User Interface Strategies for Effective Human-Computer Interaction. Reading: Addison-Wesley, 1987.

/SIB 86/ SIBERT, J. et al. An Object-Oriented User Interface Management System. ACM Computer Graphics, New York, v. 20, n. 4, p. 259-68, Aug. 1986.

/SIG 86/ SIGPLAN Notices, New York, v. 21, n. 11, Nov. 1986.

/SMA 89/ SMART, J.; LEYGUES, F.; LICHTENHEIN, M. TC33 Technical Assessment Ad-Hoc Group: User Interface. Paris: European Computer Manufacturers Association, 1989. User Interface Technical Assessment Report.

/SMI 77/ SMITH,J.M. ; SMITH,D.C.P. Database Abstractions: Aggregation and Generalization. ACM Transactions on Database Systems, New York, v. 2, n. 2, p. 105-133, June 1977.

/SMI 83/ SMITH, D.C. et al. Designing the Star User Interface. In: INTEGRATED Interactive Computing Systems. Amsterdam, North-Holland, 1983.

/SPP 90/ SCHUBERT, E.G.; PIMENTA, M.S.; PRICE, R.T. GRAEDIUS: Uma Proposta para Definição de Interfaces com o Usuário usando Gramática de Atributos. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 4., 1990, Águas de São Pedro. Anais... Sao Paulo: SBC , 1990. 280 p. p. 84-95.

/SZC 88a/ SZCZUR, M.R.; MILLER, P. Transportable Applications Environment (TAE) Plus: Experiences in Objectively Modernizing a User Interface Environment. In: OBJECT ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS CONFERENCE, Oslo, 1988. Proceedings... New York: ACM, 1989. p 58-70. OOPSLA'88

/SZC 88b/ SZCZUR, M. Transportable Applications Environment - An Integrated Design-to-Production UIMS. In: ANNUAL CONFERENCE AND EXPOSITION TO COMPUTER GRAPHICS APPLICATIONS, 9., Mar. 1988. Proceedings... p.1-17.

/SZE 88/ SZEKELY,P. & MYERS,B.A. A User Interface Toolkit Based On Graphical Objects and Constraints. In: OBJECT ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS CONFERENCE, Oslo, 1988. Proceedings... New York: ACM, 1989. p 36-45. OOPSLA'88

/TAK 90/ TAKAHASHI, T. ;LIESENBERG, H.K.E.; XAVIER, D.T. Programação Orientada a Objetos: uma visão integrada do paradigma de objetos., São Paulo, IME-USP, 1990.

/TES 81/ TESLER, L. The Smalltalk Environment. BYTE, Peterborough, v. 6, n. 8, p. 90-147, Aug. 1981.

/VAL 89/ VALDÉS, R. A Virtual Toolkit For Windows and the Mac. BYTE, Peterborough, v. 14, n. 3, p. 209-16, Mar. 1989.

/WEB 86/ WEBER, R.F. ; WEBER,T.S. Gerenciadores de Ambientes Gráficos: uma revolução na interface com o usuário, 1986.(Comunicação Pessoal)

/WIL 83/ WILSON, G.A. et al. The Multipurpose Presentation System. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 9., 1983, Florence 1983. Proceedings... Florence. VLDB Endowment, 1983. 416 p. p. 56-69.

/YOU 88/ YOUNG, M. et al. Software Environment Architectures and User Interface Facilities. IEEE Transactions on Software Engineering, New York, v. 14, n. 6, p. 697-708, June 1988.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"Um modelo canônico de ferramenta para desenvolvimento de interfaces com o usuário".

Dissertação apresentada aos Srs.

Prof. Dr. Carlos Alberto Heuser

Prof. Dr. Raul Fernando Weber

Prof. Dr. Roberto Tom Price

Prof. Dr. Rubens Nascimento Melo (PUC/RJ)

Visto e permitida a impressão

Porto Alegre, 5 / 9 / 91

Prof. Dr. Carlos Alberto Heuser
Orientador

Prof. Dr. Ricardo Augusto da L. Reis
Coordenador do Curso de Pós-Graduação
em Ciência da Computação

ESR/CCB

\DIVCPGCC\ASSINA.BAN