Universidade Federal da Paraíba - Campus II
Centro de Ciências e Tecnologia - CCT
Departamento de Sistemas e Computação - DSC
Coordenação de Pós-Graduação em Informática - COPIN

# FERRAMENTA PARA AVALIAÇÃO DE INTERFACES EM AMBIENTE WINDOWS, A PARTIR DA MONITORAÇÃO DE DADOS - $FAI_{WIN}$

**Eliane da Silva Alcoforado Diniz**

Campina Grande - PB
Agosto de 1996

**Eliane da Silva Alcoforado Diniz**

# FERRAMENTA PARA AVALIAÇÃO DE INTERFACES EM AMBIENTE WINDOWS, A PARTIR DA MONITORAÇÃO DE DADOS - $FAI_{WIN}$

Dissertação apresentada ao Curso de Mestrado em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do grau de mestre.

Área de Concentração: **Ciência da Computação.**
Linha de Pesquisa: **Engenharia de Software.**

Orientador (a): **Profª . Drª. Maria de Fátima Queiroz Vieira Turnell**

Campina Grande - PB
Agosto de 1996


D585f Diniz, Eliane da Silva Alcoforado.
Ferramenta para avaliacao de interfaces em ambiente windows, a partir da monitoracao de dados - FAI win / Eliane da Silva Alcoforado Diniz. - Campina Grande, 1996.
119 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Ferramentas de Programacao. 2. Windows - Ferramenta para Avaliacao de Iinterfaces. 3. Engenharia de Software. 4. Dissertacao - Informatica. I. Turnell, Maria de Fatima Queiroz Vieira. II. Universidade Federal da Paraiba - Campina Grande (PB). III. Título

CDU 004.4'23(043)

**FERRAMENTA PARA AVALIAÇÃO DE INTERFACES EM AMBIENTE WINDOWS, A PARTIR DA MONITORAÇÃO DE DADOS - $FAI_{WIN}$**

**ELIANE DA SILVA ALCOFORADO DINIZ**

**DISSERTAÇÃO APROVADA EM 02.08.96**

Fatima Vieira Turnell

**PROFª MARIA DE FÁTIMA Q. V. TURNELL, Ph.D**
**Presidente**

**PROF. JORGE CESAR A. DE FIGUEIREDO, Dr.**
**Examinador**

**PROF. WALTER CYBIS, Dr.**
**Examinador**

**CAMPINA GRANDE - PB**

# DEDICATÓRIA

Dedico este trabalho as minhas filhas, Tassiana e Juliana , esperando que ele sirva de incentivo durante o transcorrer de toda as suas vidas, e ao meu marido por tudo que ele representa para mim.

# Agradecimentos

A Deus, este pai misericordioso, que sempre me acompanhou e que nos momentos mais difíceis da minha vida, em que eu me sentia debilitada, carregou-me em seus braços para que eu pudesse recompor as minhas forças.

As minhas duas filhas Tassiana (12 anos) e Juliana (10 anos), que são as minhas maiores realizações, pelo carinho, amor e compreensão e por terem conseguido suportar a minha ausência, sempre incentivando-me a continuar e a concluir este trabalho.

A Gilberto Barbosa Diniz, meu marido, por ser esta pessoa maravilhosa e compreensiva, que tanta confiança deposita em mim. Pela ajuda ilimitada que sempre me prestou, pela sua dedicação as nossas filhas, e pelo seu esforço grandioso em tentar suprir a minha ausência durante o transcorrer deste trabalho.

A profª Drª. Maria de Fátima Q. V. Turnell, pela orientação, apoio, persistência e dinamismo, pois sem a sua inestimável ajuda não teria sido possível a conclusão deste trabalho e, principalmente, pela amizade inestimável e confiança em mim depositada.

A Iracema da Silva Alcoforado, minha mãe, pela sua grandiosa sabedoria que sempre nos mostrou o caminho que deveríamos percorrer e incansavelmente nos incentivando para que conseguíssemos completar o percurso e na sua sábia simplicidade se realizando através das nossas conquistas.

Aos meus irmãos e irmãs pelo apoio e estímulo que tanto me incentivaram.

Aos membros da banca examinadora, pela gentileza em participar deste processo de avaliação.

A minha amiga D. Zenir por ter tomado conta das minhas adoradas filhas para que eu pudesse, nestes quase cinco mil quilômetros que nos separam (C. Grande, PB - Pelotas, RS), ter condições de concentra-me no meu trabalho.

As minhas amigas Maria de Fátima Fernandes e Neuman Fernandes por seu carinho e acolhimento quando da minha estada em sua residência e pela grande amizade que nos une.

A Rinaldo Santos Júnior pela ajuda técnica de grande valia, durante o desenvolvimento deste trabalho e pela grande amizade e incentivo.

A André Serrano Santos, pela ajuda significativa durante o processo de desenvolvimento da interface da ferramenta utilizada neste trabalho e por sua grande amizade.

A minha querida turma, composta dos amigos: Joseana Fechine, Kíssia Carvalho, Kátia Galdino, Marckson de Souza, Paulo Márcio Passos, Iana Daya C. F. Passos, José Eustáquio Queiroz e Suely Fernandes pelo apoio incondicional que tanto me incentivou a continuar esta jornada.

A Maria de Fátima Nascimento, secretária do curso de bacharelado em Sistemas e Computação por seu incentivo e amizade.

Aos funcionários do Departamento de Engenharia Elétrica da UFPB, em especial a Angela de L. Ribeiro Matias pela amizade e pela cordial atenção com que sempre pude contar e a todos os funcionários lotados no Laboratório de Processamento de Sinais (LAPS) por sua atenção e apoio técnico que foram imprescindíveis para a conclusão deste trabalho.

Aos funcionários do Departamento de Sistemas e Computação da UFPB, em especial a Ana Lúcia Guimarães, pela eficiência e presteza, com as quais pude contar durante todo o curso de pós-graduação.

Em memória a
João Guedes Alcoforado, meu pai, que sempre desejou ver seus sonhos realizados através dos seus filhos, a minha eterna gratidão e aonde o senhor estiver receba este título para aumentar a sua galeria de realizações.

# Índice de Figuras

# Índice de Tabelas

# Resumo

Este trabalho apresenta uma ferramenta para avaliação de interfaces de aplicações desenvolvidas para ambiente Windows. O método de avaliação utilizado é do tipo empírico e a técnica empregada é a monitoração dos dados. Os dados são coletados em duas etapas. Na primeira, a interface é analisada do ponto de vista das definições de projeto, abrangendo informações relativas ao projeto de tela, de diálogos, etc.. A segunda etapa monitora a interação com o usuário, registrando seu desempenho quanto a incidência de erros, solicitação de ajuda, tempo para execução de tarefas, etc. Os dados coletados durante a interação são armazenados em um arquivo para posterior análise. A ferramenta oferece ainda recursos para apresentação dos dados coletados de modo a facilitar a sua análise. O trabalho conclui com a apresentação de um estudo de caso, e a análise dos resultados obtidos pela ferramenta.

# Abstract

This works presents an interface evaluation tool developed to be used with windows applications. The evaluation method is empirical and the technique employed by the tool is data monitoring. The data is collected in two phases. In the first phase, the interface is analysed from the viewpoint of the interface design definitions, such as: screen design, dialogue design, etc. The second phase monitors the user interaction, registering his performance in respect to: error rate, help request, task execution time, etc. The data collected during the interaction session is stored in a log file for later analysis. This evaluation tool offers facilities to present the data in an easy way for analysis. Finally it presents a case study with the analysis of the tools' results.

# *Capitulo I*

# *Introdução*

Após décadas de preocupação voltada para o desenvolvimento e o aperfeiçoamento da tecnologia computacional, as tendências da ciência da computação se voltam para estratégias de projetos de sistemas que realmente se adequem ao usuário. Até então, vinha sendo dada pouca importância à interface Homem-Máquina (HM), colocando-a em um plano secundário. Tal evolução advém do amadurecimento de campos tradicionais da tecnologia correlata a ciência da computação.

A Partir do inicio da década de 90, dentre os vários segmentos da ciência da computação, nenhum teve mais alterações do que, aquele que trata da interação HM. Isto decorreu do desenvolvimento da tecnologia de *software*, atualmente voltada para os ambientes baseados em microcomputadores. Estes ambientes por razões de custo e transformações tecnológicas, vem atingindo um público diversificado, deixando de ser de uso exclusivo das áreas de ciência e tecnologia.

A partir deste fato surgiu a necessidade de melhorar a interação entre o *software* e o usuário final, o qual passou a ser um elemento chave nos projetos de interface HM. O desenvolvimento de novas técnicas, a adesão de novas comunidades de usuários das mais diversas formações e a proposta de novas tarefas, fez com que a interface apresentada ao usuário viesse a se tornar um dos fatores mais significativos na qualidade de um sistema computacional interativo.

A eficiência do processo interativo HM se traduz em uma série de fatores: a) na funcionalidade da comunicação entre o usuário e a grande variedade de facilidades disponíveis; b) na segurança que o sistema transmite ao usuário ao executar suas tarefas, permitindo-lhe prever o que ocorre após cada uma das suas ações; c) na integridade das informações fornecidas ao usuário; enfim, na relação tempo/custo computacional exigido para a execução de uma tarefa.

Recursos gráficos interativos são atualmente um dos meios mais "naturais" de comunicação HM, já que a computação gráfica oferece condições de apresentação espacial da informação como alternativa para a informação textual [MYER80]. A taxa de transferência de informação, o acesso e a decodificação do objeto da comunicação são muito mais eficientes para a informação gráfica do que para a textual, já que o sistema visual humano é considerado como um sensor extremamente eficiente para o processamento de imagens em tempo real [RAED85].

Surgem diariamente, dispositivos de interface dotados de recursos gráficos cada vez mais eficientes e ergonômicos, tornando possível a implementação de interfaces mais sofisticadas. Aplicações criadas a partir de princípios e processos ergonômicos surgem inovando o mercado consumidor com recursos muito mais eficientes do que aqueles oferecidos pelas aplicações convencionais [SHNE92].

O advento dos sistemas computacionais com estas características não representam, por si só, uma solução para a especificação e a implementação de interfaces eficientes. Estes recursos constituem apenas suportes físicos para projetos de ambientes interativos. Dentre estes recursos destacamos as ferramentas de prototipagem e desenvolvimento rápido oferecidos pelos novos pacotes de gerenciamento de diálogos. Estas, vêm permitir que os projetistas diminuam sensivelmente a sua preocupação com os detalhes de implementação e passem a concentrar sua atenção na qualidade do produto em construção.

Todavia, nem mesmo estes recursos são suficientes para a concepção de processos interativos satisfatórios. O conhecimento mais aprofundado do usuário vem complementar os recursos necessários ao desenvolvimento de interfaces. A psicologia cognitiva vem desempenhando um papel cada vez mais relevante no processo de modelagem dos usuários, ao considerar suas diferenças individuais [NORC89]. Por outro lado, a ergonomia vem sendo atribuída de um valor sócio-econômico cada vez mais significativo[MUNI79], já que sua meta principal é a otimização da relação entre o usuário e o sistema que o auxilia [SOND82].

O sucesso de qualquer produto que abranja a interação HM, está implicitamente ligado à satisfação do usuário, a qual se encontra fortemente atrelada à sua concepção do produto e à sua avaliação da natureza da aplicação considerada. Daí, sem uma estrutura

adequada de especificação, implementação e avaliação da interface proposta para um sistema específico, muito da sua funcionalidade poderá vir a ser comprometida.

É necessário portanto adequar os recursos de qualquer projeto aos seus usuários potenciais, observando suas diferenças individuais segundo critérios relacionados com a formação acadêmica, a freqüência de uso, capacidade psicofisiologicas e cognitivas, bem como com base nos aspectos ambientais [SHNE92]. Para isto, se faz necessário que todo projeto de Interface com o Usuário(IU) seja avaliado.

## *1.1. Avaliação de Interfaces*

A avaliação de interfaces deve utilizar métodos que envolvam vários critérios de aceitação. Dentro estes critérios pode-se citar: tempo de aprendizagem de funções específicas; velocidade para realização de uma dada tarefa; taxa de erros; taxa de solicitação de ajuda, satisfação do usuário; retenção humana de comandos durante à interação; a distribuição espacial dos objetos em uma dada janela, entre outros.

A maioria da técnicas de avaliação encontradas na literatura [LEA88, DOWN92, TREU94], aborda os aspectos qualitativos da interface avaliada. Mesmo algumas técnicas como por exemplo, os questionários, que podem ser analisados quantitativamente, se baseiam em informações heurísticas e portanto deixam de levar em consideração os aspectos objetivos que podem influir grandemente na qualidade das interfaces. Uma avaliação para ser completa deve levar em consideração ambos as abordagens: quantitativa e qualitativa. No entanto faltam ferramentas para auxiliar na avaliação quantitativa, permitindo mensurar parâmetros que possam servir para uma avaliação objetiva da qualidade das interfaces.

Os métodos de avaliação requerem medidas e comparações frente aos resultados esperados [COX93]. A avaliação no sentido mais amplo, inclui muitas técnicas diferentes, as quais podem ser usadas individualmente ou combinadas. Entre as várias técnicas de avaliação podemos destacar: Observação; Comentários; Gravação de vídeo; Questionário; Entrevista, etc. [SHNE92 ,LEA88 ,DOWN92].

Apesar de suas diferenças todas estas técnicas tem como meta principal auxiliar o projetista de interfaces na localização de problemas de modo que, a próxima versão da interface leve ao aumento da produtividade, e da satisfação do usuário.

## *1.2. O Estado da Arte*

A necessidade de avaliação e validação das interface HM, tem feito surgir novas técnicas para avaliar o grau de satisfação do usuário com os sistemas por eles utilizados. Como já foi observado, a maioria das técnicas de avaliação (aplicações de questionários; comentários; gravação de vídeo, entrevistas, etc.) disponíveis têm se concentrado em uma avaliação subjetiva da interface com base no grau de satisfação do usuário.

Apesar da crescente preocupação dos projetistas quanto à qualidade das interfaces, pouco se conhece na literatura da existência de ferramentas para sua avaliação. Há, no entanto, um grande número de experiências publicadas quanto à avaliação qualitativa destas interfaces. No maioria dos casos esta avaliação tem um caracter subjetivo pois se baseia na opinião dos usuários sobre uma interface e não na avaliação do desempenho na realização de tarefas através do uso desta interface. Por outro lado, a avaliação quantitativa se baseia na coleta de dados sobre a interação Usuário-computador.

Um exemplo de sistema que coleta dados de interação é o Telecom Gold [BOUC92]. No entanto o propósito de sua coleta é a adaptação da interface em tempo real, ao perfil e desempenho do usuário.

Dentre as ferramentas que tem o propósito de coletar dados durante a interação do usuário com a interface destacamos a ferramenta ***Playback.*** [NEAL84]. Esta ferramenta suporta uma técnica de avaliação desenvolvida no Centro de Fatores Humanos da IBM, que tem o propósito de avaliar a *usabilidade* do *software* e de sua documentação. A idéia central do ***Playback*** é capturar as informações durante uma sessão de interação do usuário com o produto que está sendo avaliado.

Nesta abordagem, um computador intercepta, grava e registra o tempo de acionamento de teclas. A análise é realizada a partir do arquivo de dados, que contém a transcrição da sessão. O arquivo gerado eqüivale a uma gravação em vídeo, embora não permita associação direta dos eventos com comentários e observações.

A ferramenta ***Playback*** coleta e fornece as seguintes informações:

- Freqüência de ocorrência de incidentes críticos;
- Intervalo de tempo de tais incidentes;

- Freqüência de uso dos comandos e teclas de funções;
- Tempo gasto na ajuda;
- Tempo de uma sessão e;
- Número de solicitação de ajuda.

Segundo os autores da metodologia, esta ferramenta é muito eficiente na avaliação objetiva da Interface, embora ainda demande a revisão manual da transcrição.

Sioch em [SIOC191] descreve uma ferramenta, que tem como objetivo gravar a transcrição automática dos dados durante uma sessão do usuário, *a MAXIMAL REPEATING PATTERNS (MRP).* A transcrição de uma sessão com o usuário é um registro completo das ações de entrada do usuário e das respostas geradas pelo sistema como resultado de sua utilização. As ações do usuário são extraídas desta transcrição e representam uma seqüência das entradas do usuário ordenadas no tempo.

Esta ferramenta está limitada em seu uso a interfaces do tipo linha de comando. Seu objetivo é buscar um padrão de repetição de uma seqüência de comandos na transcrição de uma sessão com o usuário. A técnica associada à ferramenta se baseia na hipótese de que, a repetição de ações do usuário é um indicador em potencial de problemas na interface.

Cohill e Enrich em [COHI83], descrevem um conjunto de programas e rotinas desenvolvidas para coletar dados do teclado e informações de estado e comprimi-las. As variáveis medidas são:

- Tempo gasto na ajuda;
- Número de vezes de solicitação de ajuda e,
- Freqüência no uso de comandos.

Essa ferramenta insere código em pontos estratégicos do sistema que está sendo investigado.

Os autores recomendam coletar tanto dados quanto possível, ao invés de uma coleta seletiva, apesar da grande quantidade de dados envolvidos.

Hanson e Kraut em [HANS84], Coletaram dados sobre comandos do sistema *UNIX* e aplicaram técnicas de análise estatística aos dados. Eles determinaram um núcleo dentro do conjunto de comandos, a partir do estudo da freqüência de comandos. Como conseqüência deste trabalho, prescreveram uma restruturação da interface do *UNIX,* baseados nestes resultados.

Uma outra ferramenta para coleta de dados de uma interação é apresentada em [NAKA95]. Esta ferramenta, FAI- ferramenta de avaliação de interface, se propõe a avaliar quantitativamente uma interface gerada no ambiente AGILE [PROC92, SANT92]. Nela, os dados decorrentes da interação HM são coletados e, a partir deles é gerado um relatório da ocorrência, complementando os tradicionais métodos de avaliação qualitativa. A FAI foi desenvolvida na linguagem de programação "C" para plataforma MS-DOS, em computadores do tipo IBM-PC. A principal limitação desta ferramenta é estar restrita a avaliações de interfaces geradas para o ambiente AGILE.

## *1.3. Motivação para o Trabalho*

Motivado pela carência de ferramentas para avaliação quantitativa de interfaces este trabalho se propõe a contribuir com o desenvolvimento de uma ferramenta de avaliação das interfaces prototipadas para sistemas em ambiente *Windows*, a $FAI_{WIN}$.

A principal razão para escolha deste ambiente foi a espectativa de repercussão do trabalho levando-se em consideração a popularidade deste ambiente.

Considerando o grande número de ferramentas de prototipagem de interfaces para aplicações desenvolvidas para este ambiente, tais como: *Resource Workshop*, o *Protoview/Protogen, Microsoft Visual Basic, Microsoft Visual C, Delphi* entre outras, e considerando ainda que apesar deste ambiente ser considerado um padrão de interação; muitas alternativas de projeto são deixadas a cargo do projetista, este trabalho propõe uma ferramenta para avaliação de interfaces desenvolvidas para este ambiente.

## *1.4 Objetivos do Trabalho*

Como pode ser observado, as ferramentas de avaliação quantitativa descritas neste capítulo, estão restritas a ambientes específicos. Esta limitação leva a um grande esforço de desenvolvimento de ferramentas para atender a um grande número de ambientes de interação. Um outro problema comum às técnicas de coleta de dados é a necessidade de inserir modificações nas interfaces sob avaliação ou ainda de desenvolver um *hardware* (externo) para a coleta de informações dos dispositivos de interação como o teclado.

Diante destas limitações, o desenvolvimento desta ferramenta se propõe a atingir os seguintes objetivos:

- auxiliar na avaliação de diferentes aspectos das interfaces prototipadas para o ambiente *Windows*;
- fornecer dados de forma clara e objetiva, para o projetista
- assegurar a generalidade da ferramenta de modo a poder avaliar interfaces geradas para ambiente *Windows* por diferentes ferramentas.

## *1.5 Apresentação do Texto*

Este trabalho apresenta, no capítulo II, os Prototipadores de Interface com o Usuário, fazendo uma breve descrição da sua evolução, e apresentando os principais prototipadores para o ambiente *Windows*, bem como considerações sobre seu uso.

O capítulo III descreve a importância da avaliação, incluindo os tipos de avaliação, os aspectos a serem avaliados, as fases do ciclo de vida do produto em que deverá ocorrer a avaliação e as principais técnicas de avaliação utilizadas. Chamando a atenção do leitor para a necessidade da avaliação de um produto de interação Homem-Máquina.

O capítulo IV aborda o escopo, a arquitetura e a descrição funcional da ferramenta de avaliação que está sendo desenvolvida. Bem como detalhes genéricos da implementação dos módulos que compõem a ferramenta.

O capítulo V apresenta um estudo de caso, onde se procura ilustrar o uso da ferramenta desenvolvida, comentando os resultados obtidos e como poderão ser utilizados pelos avaliadores.

O capítulo VI procura fornecer ao leitor uma visão geral do trabalho desenvolvido, através da discussão dos objetivos atingidos, apresentando as limitações da ferramenta, as contribuições esperadas e sugestões para o seu refinamento e continuidade.

Finalmente, os apêndices A e B trazem uma descrição de dois recursos de programação empregados nesse trabalho - DLL e *HOOK* respectivamente.

# Capítulo II
# Prototipadores de Interfaces com o Usuário

Os sistemas interativos gráficos, em sua maioria, permitem a manipulação direta de objetos gráficos e por conseguinte são reconhecidamente mais eficientes do que os sistemas interativos convencionais baseados em texto [SHNE83]. Apesar desta grande vantagem, a criação de interface para os ambientes gráficos nem sempre foi uma tarefa muito fácil para os programadores. Esta dificuldade decorria principalmente da falta de ferramentas apropriadas que facilitassem os seus trabalhos.

## 2.1 Evolução das Ferramentas de Prototipagem de Interfaces

Nos últimos anos um grande número de ferramentas para criação de interfaces com o usuário, vêm sendo desenvolvida. Infelizmente, as primeiras ferramentas eram difíceis de usar. Seus manuais eram numerosos e continham centenas de procedimentos. Algumas demandavam que os programadores aprendessem uma nova linguagem de programação para que pudessem criar suas interfaces. Como dificuldade adicional exigiam o uso de pacotes gráficos adjacentes, para que figuras geométricas (a exemplo de ícones) pudessem ser desenhadas. Uma vez que o olho humano é sensível a pequenas diferenças, o *display* gráfico deve atingir elevados níveis de qualidade, um simples erro no alinhamento de um *pixel* poderá ser visível e a maioria dos pacotes gráficos disponíveis então não fornecia ajuda na construção de *display* atrativos. A despeito da complexidade no uso daquelas ferramentas, elas não ofereciam flexibilidade suficiente para produzir os efeitos desejados nos projetos.

Uma das formas de tonar mais fácil a criação e manutenção de um *software* é assegurar a modularidade das diferentes partes que o compõem. A maioria das ferramentas de prototipagem de interface garante a independência da interface em relação ao *software* aplicativo, permitindo assim que a interface possa ser modificada

sem afetar o código da aplicação. Infelizmente, os programadores na prática encontram dificuldades ou até a impossibilidade em separar a interface da aplicação, e as mudanças na interface usualmente implicam em reprogramação da aplicação. Além disso, os atuais *kits* para construção de interface agravam este problema com constantes chamadas a procedimentos, que se encontram encapsulados nas aplicações. Usualmente, cada objeto (menu, barra de rolamento, caixa de texto, etc.) que se encontra sobre a janela, requer que o programador indique pelo menos um procedimento na aplicação, que deve ser chamado quando o usuário o ativar. Uma vez que uma interface pode ser composta de vários objetos haverá vários procedimentos correspondentes na aplicação o que cria um enorme esforço de manutenção. Como conseqüência os projetistas tem que passar por um rigoroso treinamento de modo que na condição de especialistas nas ferramentas produzam uma melhora significativa na qualidade dos projetos de interfaces além de continuar demandando o envolvimento de outros profissionais como projetistas gráficos e redatores técnicos.

A dificuldade existente na implementação de interfaces, que utilizam ferramentas, onde há uma grande quantidade de elementos a serem programados, estão começando a serem minimizadas com um novo grupo de prototipadores que vêm sendo desenvolvidos para facilitar estas tarefas.

Atualmente, as ferramentas gráficas para construção de interfaces têm ganho popularidade por oferecerem recursos que se contrapõem àqueles das linguagem convencionais - é a programação visual. Estes recursos simplificam a criação de aplicações gráficas altamente interativas. Estas ferramentas reduziram o esforço exigido para a criação de aplicação, que envolvam: animação dos objetos sobre a janela, pacotes de desenhos, sistema de visualização de dados e programas, linguagem de programação visual, e jogos [ZNAD94].

Existe um determinado grupo de ferramentas - "construtores de interface", para várias plataformas tais como: o *HiperCard* da *Apple*, *Visual Basic* e *Visual C++* da *Microsoft*, *Delphi* da *Borland*, *Guide* da *Sun*, *NewWave* da *Hewlett-Packard*, etc., que tornam a prototipagem e criação de interfaces uma tarefa bem mais fácil. De fato, os *kits* de ferramentas de interface com o usuário, são um dos exemplos de *reusabilidade*; independência de plataforma; e de bibliotecas portáveis.

As ferramentas interativas de interface têm demonstrado um aumento na produtividade do programador, e estão ajudando os projetistas a criarem interfaces melhores.

A seguir serão apresentadas algumas ferramentas de prototipagem de interface para o ambiente *Windows*, já que este trabalho é voltado para este ambiente.

## 2.2 *Ferramentas de Prototipagem de Interfaces para Ambiente Windows*

Dentre as várias ferramentas de prototipagem para o ambiente *Windows*, serão enfatizadas neste tópico as ferramentas *Visual Basic* e *Visual C++* da *Microsoft* e a ferramenta *Delphi* da *Borland,* por serem essas as abordadas ao longo do desenvolvimento deste trabalho. Nesta sessão se fará uma breve explanação da funcionalidade destas ferramentas, ressaltando seus pontos positivos e mencionado algumas limitações encontradas durante as suas utilizações.

### 2.2.1 *Visual Basic*

O *Visual Basic* para o *Windows* da *Microsoft* é um recurso estimulante para os projetistas que tencionam escrever aplicativos para o *Windows*. Dotado de um mecanismo de programação orientado a eventos é uma ferramenta que suporta o projeto visual. Utilizando-se do ambiente gráfico do *Windows* para a construção rápida de aplicativos poderosos, o *Visual Basic* torna transparente a complexidade da programação para o ambiente *Windows*. A combinação de recursos já existentes na linguagem *Basic* com ferramentas de projeto visual, introduz simplicidade e facilidade de uso, sem sacrificar o desempenho ou as características gráficas que caracterizam o *Windows* como um ambiente agradável de trabalhar. Menus, fontes, caixas de diálogo, campos de texto com deslocadores e outros recursos, podem ser facilmente projetados e controlados, demandando apenas algumas poucas linhas de programação[NELS94].

O *Visual Basic* torna o trabalho dos projetistas mais produtivo, fornecendo uma ferramenta que abrange diferentes aspectos do desenvolvimento da Interface Gráfica do Usuário - GUI. A criação de uma interface gráfica é feita através do desenho dos seus objetos em meio gráfico. Durante a fase de criação as propriedades de cada objeto são

definidas determinando sua aparência e comportamento. Em seguida deve ser escrito o código que responderá aos eventos que ocorrerão na interface durante a interação com o usuário [MICR93A].

Com o *Visual Basic* o projetista pode criar interfaces capazes de explorar características do *Microsoft Windows* tais como: a Interface de Documentos Múltiplo - MDI[1] (*Multiple-Document Interface*), Objetos Ligados e Aninhados - OLE[2] (*Object Linking and Embedding*), Intercâmbio Dinâmico de Dados - DDE[3] (*Dynamic Data Exchange*), gráficos, entre outras. O *Visual Basic* pode ser estendido pela adição de controles personalizados e chamadas a procedimentos em biblioteca de ligação dinâmica -DLL[4] (*Dynamic Link Library*, ver Apêndice A), que possibilitam o seu interfaceamento com outras linguagens de programação. Portanto é possível utilizar rotinas prontas escritas em outras linguagem permitindo a reusabilidade de código.

### 2.2.2 *Visual C++*

O *Visual C++* da *Microsoft* é uma ferramenta de construção e depuração de aplicações e bibliotecas em um ambiente integrado *Windows*. O *Visual C++* torna muito mais fácil o trabalho complexo de desenvolvimento de aplicação para o ambiente *Windows*, a partir da incorporação das classes de alto nível das ferramentas de desenvolvimento, hospedeiras do *Windows*.

Os componentes do *Visual C++* incluem:

- *Visual Workbench*
- Sistema de compilação C/C++
- Biblioteca de Classe
- Editor de Recurso *App Studio*
- Bibliotecas de Tempo de Execução

---

[1] Interface de Documentos Múltiplo - MDI (*Multiple-Document Interface*) é uma especificação para aplicativos que lidam com documentos no ambiente *Microsoft Windows*. A especificação descreve uma estrutura de janela e de interface que permite ao usuário trabalhar com múltiplos documentos dentro de um único aplicativo.

[2] Objetos ligados e aninhados - OLE (*Object Linking and Embedding*) é um protocolo para compartilhamento de dados entre aplicações desenvolvidas pela *Microsoft*.

[3] Intercâmbio Dinâmico de Dados - DDE (*Dynamic Data Exchange*) é um protocolo usado para compartilhamento de dados entre aplicações originalmente desenvolvidas para ambiente *Windows*.

[4] Biblioteca de ligação dinâmica -DLL (*Dynamic Link Library*) é uma biblioteca compartilhada ou de carga dinâmica na terminologia *Windows*.

- Exemplo de Código Fonte
- Arquivo de Ajuda *On-line*
- Ferramentas de construção
- Soluções de depuração

Dentre estes componentes serão destacadas as ferramentas: *Visual Workbench, App Studio, AppWizard, ClassWizard*, e vários outros utilitários utilizados a partir do *Visual Workbench* ou por ele usados.

Durante o processo de desenvolvimento, o *AppWizard* é usado para criar arquivos fontes da biblioteca de classes do C++ da *Microsoft*. Estes arquivos serão utilizados no projeto em desenvolvimento. Para criar e editar recursos tais como: caixas de diálogos, formulários, menus, barras de ferramentas, e controles, deve-se usar o *App Studio*. Utiliza-se a ferramenta *ClassWizard* para: criar novas classes, adicionar o código C++ aos mapas de mensagens, ligar os dados, entre outras funções.

Entre os componentes acima relacionados, pode-se destacar o *Visual Workbench*, pois este componente é a base da plataforma de desenvolvimento do *Visual C++*. Ele contém: um editor de texto com sintaxe sensível a cor, um *browser* gráfico, um compilador, um *linker*, um depurador, um utilitário *make*, e uma ajuda *on-line* integrada. O *Visual Workbench* é o ponto central de uma estratégia de desenvolvimento a partir do qual todas as outras atividades de desenvolvimento são realizadas.

O *Visual C++* inclui ainda os controles personalizados .VBX (*Visual Basic Extended*), e rapidamente desenha e edita bitmaps, cursores, ícones, e barra de ferramenta, além de criar novos arquivos de recursos e abrir e editar arquivos de recursos já existentes. A sua biblioteca de tempo de execução contém mais de 550 funções e macros projetadas para serem usadas com aplicações C/C++. Estas funções incluem entrada/saída, alocação de memória, suporte matemático, controle de processo, gráficos e muitos outros.

### *2.2.3 Delphi*

O *Delphi* é uma das mais recentes ferramentas no campo da programação visual. O *Delphi* é descendente da linguagem Pascal e possui uma interface visual que elimina esforços desnecessários durante a sua utilização.

Ele oferece vários componentes da programação de interfaces, previamente implementados. Alguns são simples, como botões, caixa de seleção e rótulos, outros incorporam elevada complexidade, como a exibição de um quadro de diálogo para permitir que os usuários escolham que arquivo abrir. Estes componentes do *Delphi* são equivalentes aos arquivos .VBX do *Visual Basic*. Os componentes padrão do *Delphi* são uma combinação de elementos de tela embutidos no *Windows*.

Para aumentar a facilidade de uso do *Delphi*, ele possui uma paleta básica que é chamada de *Component*. Esta paleta possui sete páginas onde cada página contêm elementos chaves do *Delphi*. A seguir serão relacionadas estas páginas, mostrando o potencial de programação do *Delphi* que se encontra embutido em cada página:

- Padrão (*Standard*) - A maioria dos componentes desta página corresponde diretamente a elementos na tela do *Windows*. Há menus, botões de acionamento, barras de rolamento, etc.

- Adicional (*Additional*) - Esta página contém componentes mais avançados, por exemplo: o componente de estruturação, que é responsável pela exibição de dados organizados em uma espécie de hierarquia. Um outro componente é o controlador de mídia que permite que os programas emitam música, sons e vídeo. Há ainda componentes cujo objetivo principal é exibir informações gráficas.

- Controle e Acesso aos Dados (*Data Access e Data Control*) - O *Delphi* usa o *Borland DataBase Engine* (BDE) para acessar arquivos de bancos de dados, em vários formatos de arquivo. Permite a combinação dos serviços de bancos de dados fornecidos pelo BDE como: leitura, escrita, indexação e consulta multiusuário, de tabelas do *DBASE* e do *Paradox*, etc.

- Diálogos - Permite que os programas em *Delphi*, tenham fácil acesso aos diálogos comuns introduzidos pelo *Windows*. Diálogos para: operação de arquivos, seleção de fontes, seleção de cores, etc.

- Sistema - Esta página permite que sejam combinados elementos tais como diretório e listas de arquivos. Ela também contém os componentes que tratam da comunicação de alto nível entre programas via DDE e OLE..

- Amostras (*Samples*) - É uma página que reúne componentes que não são intrínsecos ao pacote do *Delphi*

Além de todas estas vantagens citadas, o *Delphi* também pode usar e criar DLLs. Os programas em *Delphi* contém as porções necessárias de suas bibliotecas de tempo de execução (*run-time).* O *Delphi*, também realiza uma tarefa completa de compilação dos programas no código de máquina que o computador compreende, aumentado assim a sua velocidade de execução.

## *2.3 Considerações sobre o Uso de Prototipadores.*

Apesar de todas as vantagens trazidas pelas ferramentas de prototipagem de interfaces HM, decorrentes da padronização dos elementos básicos para construção de interface e, da grande flexibilidade que estas ferramentas colocam à disposição dos projetistas para a sua criação, muito da aparência e funcionalidade do produto em desenvolvimento são decorrentes do bom senso do projetista. Dependendo da sua experiência e do conhecimento que tenha sobre os usuários para os quais se destina a interface projetada poderá o projetista fazer bom uso destes recursos.

Decorrente desses fatores que são inerentes a qualquer processo criativo é sempre necessário que o projeto seja submetido a uma avaliação.

# Capítulo III

# Avaliação de Interfaces com o Usuário

A área da Engenharia de *Software*, no que diz respeito ao desenvolvimento de Interfaces com o Usuário, vem recebendo mais atenção por parte dos pesquisadores. As novas técnicas de produção de *software* tem levado a uma melhora na metodologia geral de desenvolvimento das interfaces.

Segundo Desoy, Livery e Sheppard em [DESO89], a interface do usuário é especialmente importante porque ela é visível para todos os usuários do sistema, tornando-se o primeiro critério de aceitação de um sistema de *Software*.

A interface tem que ser transparente para o usuário, isto é, flexível e de fácil aprendizado. A adequação dos recursos de um projeto aos seus usuários potenciais, deve considerar critérios relacionados com a sua formação acadêmica, capacidade psicofisiologicas e cognitivas, a freqüência de uso do produto, bem como com os aspectos ambientais envolvidos [SHNE87].

A interação do usuário com o computador precisa passar por uma validação que permita averiguar o grau de usabilidade e a qualidade da interface projetada. Uma ferramenta utilizável é aquela que o usuário detém o controle da interação e obtém satisfação em utilizá-la. Na melhor das hipóteses, uma interface mal projetada resultará em custos adicionais de treinamento de pessoal e em uma grande proporção de erros na interação. Decorrente destes fatores é de fundamental importância que o seu projeto tenha passado por um rigoroso processo de avaliação. A construção de uma boa interface requer muitas revisões, avaliações e aperfeiçoamento do projeto. Portanto é de fundamental importância avaliar o produto que está sendo projetado, testando-o com diferentes tipos de usuários, de modo que possam ser detectadas as falhas de projeto.

O objetivo específico ou o foco em uma avaliação, pode estar relacionado a um ou mais objetos, os quais podem ser considerados tanto individualmente quanto combinados entre si. Segundo Treu em [TREU94] a avaliação pode tomar diferentes formas. Pode ser de natureza informal, ou pode ser sistemática e rigorosa; seus resultados podem ser subjetivos, objetivos ou ambos. Em uma avaliação os objetivos e o escopo devem estar claramente especificados. Isto significa que os aspectos da interface a serem avaliados devem ser bem definidos.

Ainda, conforme Treu em [TREU94], um método geral para avaliação de um projeto HM, está modelado na Fig. 3.1. Este método é análogo aos métodos descritos para o projeto de interfaces, mas enquanto na fase de projeto os métodos são guiados pelos princípios e fatores que causam a determinação das características da interface, na fase de avaliação os métodos são voltados para métricas e outros indicadores para avaliar a aparência, a existência, e o comportamento destas características.

O projeto de interface HM e os métodos de avaliação devem estar intimamente ligados. Esta ligação poderá ocorrer em paralelo com o processo de desenvolvimento da interface, sendo diretamente influenciado pelos resultados intermediários da avaliação, permitindo assim uma realimentação para o re-projeto.

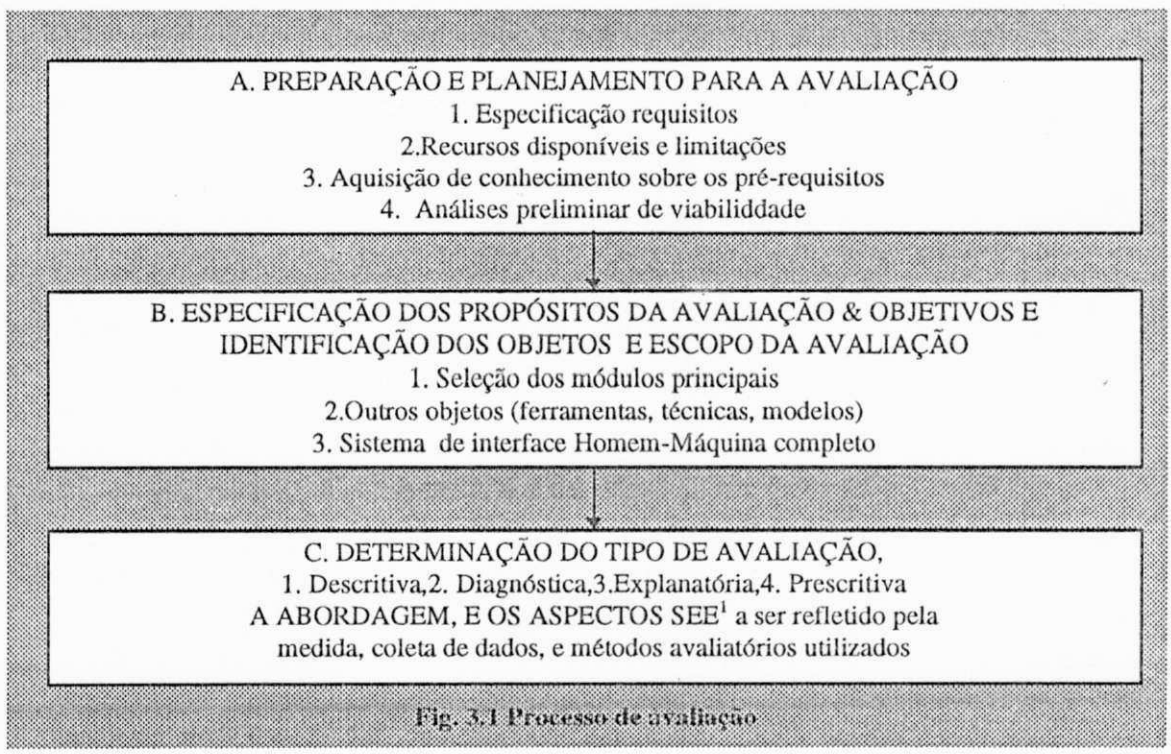



Fig. 3.1 Processo de avaliação

[1] SEE (Synergism, Efficiency,Effectiveness) - Sinergismo, Eficiência e Eficácia

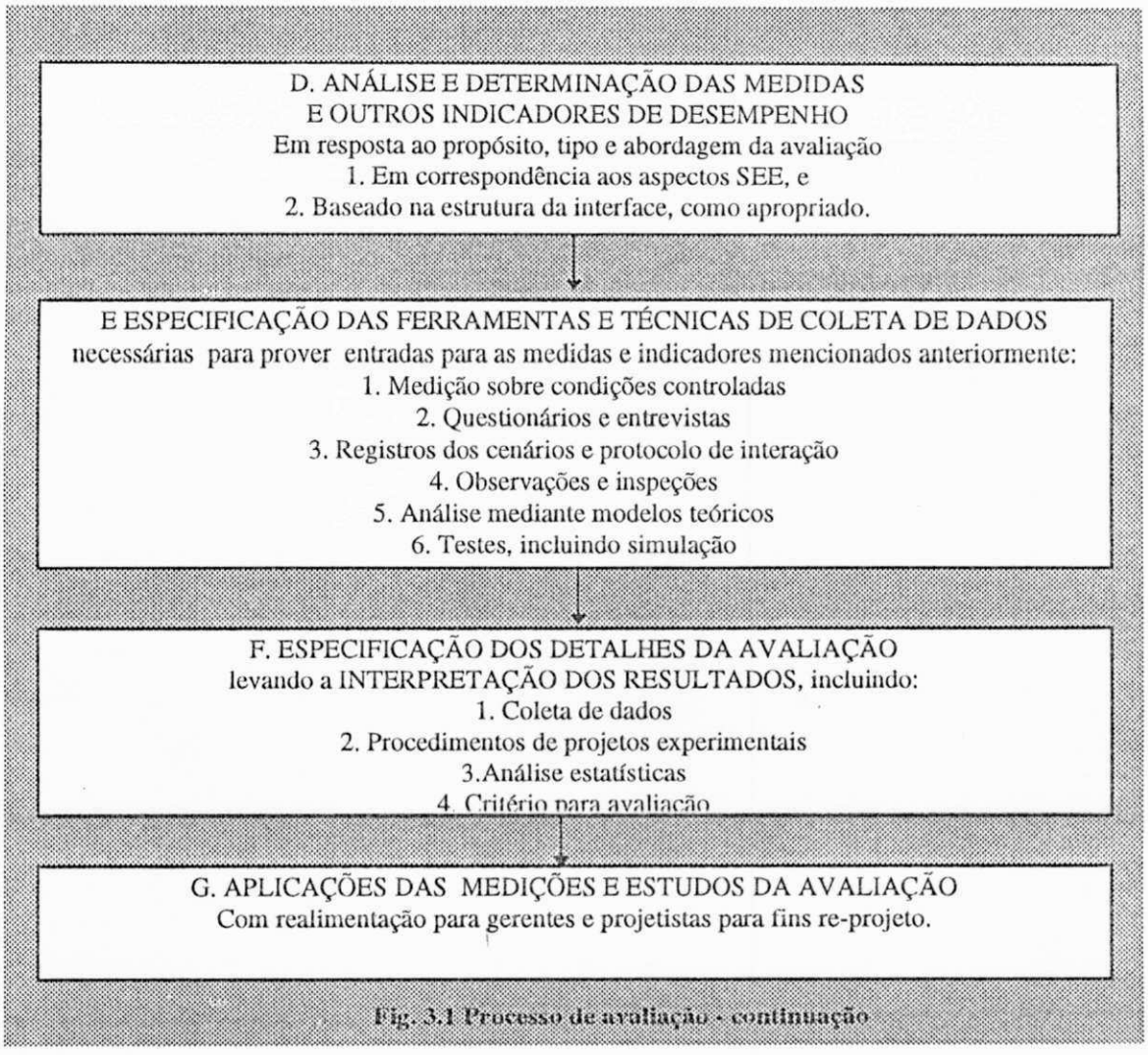



**Fig. 3.1 Processo de avaliação - continuação**

## *3.1 Importância da Avaliação.*

Nas primeiras décadas de desenvolvimento de programas para computadores, os programadores projetavam editores de texto, linguagens de controle de sistemas operacionais, linguagens de programação e pacotes de aplicação para si próprios e para seu grupo de trabalho [SHNE87].O Potencial tecnológico da época não permitia a construção de uma interface mais elaborada, restringindo-se por exemplo a *prompts* como o do DOS. Daí, o desinteresse na avaliação das interfaces geradas para os ambientes computacionais existentes naquele período.

Com o advento das interfaces gráficas e a popularidade dos microcomputadores, o estilo do passado rendeu-se à necessidade de adequar o projeto de Interface à destreza, aos anseios e à orientação do usuário [SHNE87].

## 3.2 *Tipos de Avaliação*

A elaboração dos propósitos da avaliação pode resultar em vários tipos de avaliação. Treu em [TREU94] relaciona os principais, como sendo:

- Descritivo;
- Diagnóstico;
- Explanatório;
- Prescritivo.

Para um melhor entendimento do tipo Descritivo, é apresentado um estudo sobre a produtividade de um grupo de usuários quando da interação com uma determinada interface. Neste exemplo o desempenho do usuário é o principal objetivo a ser avaliado. Comparando-se cada usuário em relação ao grupo, pode-se fazer uma *avaliação descritiva*, onde se verifica se determinado usuário teve seu desempenho "Bom" ou "Ruim" quando comparado aos outros membros do grupo em questão.

No tipo Diagnóstico é feita a observação de um determinado usuário quando de sua interação com a interface em um dado instante. Em um diagnóstico é registrado o nível de seu desempenho, informando se o usuário está interagindo da forma "Certa" ou "Errada", isto é, se ele está produzindo adequadamente.

A avaliação Explanatória consiste na investigação junto ao usuário das razões que o levaram a percorrer caminhos diferentes na interação em ocasiões distintas. Independentemente deste caminho ter sido "Melhor" ou "Pior" do que aquele caminho percorrido originalmente.

O tipo Prescritivo é especialmente utilizado para comparar alternativas de objetos ou de suas características, perscrutando os efeitos decorrentes. Neste tipo de avaliação o projetista de interface pode obter informações detalhadas sobre características da interação, de modo a apoiar as decisões sobre as alternativas de projeto da interface.

## *3.3 Aspectos a Serem Avaliados.*

Um bom projeto de interface, assim como a análise crítica de uma interface já implementada, procura se respaldar em aspectos técnicos, tais como:

- Entrada no Sistema
- Apresentação da Informação
- Modo de Comunicação Usuário-Sistema
- Documentação sobre o Sistema
- Documentação sobre a Aplicação.

Deve-se considerar também Aspectos Psicológicos envolvidos no processo de comunicação [SOND82]. Pois é unânime a opinião de que a solução de problemas relativos à usabilidade, consistência e avaliação geral de um projeto de interface pode ser facilitada a partir da consideração de fatores humanos.

O escopo da avaliação pode ficar restrito à superfície visível da interface, ou pode abranger o desempenho do usuário, o qual será considerado o objeto da avaliação. Isto significa que o avaliador deve saber exatamente o que será considerado **variável** em uma situação de avaliação e o que será mantido como **constante**, delimitando o contexto sobre o qual os objetos da avaliação estão inseridos e, definindo o que deve ser avaliado.

Se o propósito da avaliação, os objetivos específicos, e o tipo de avaliação estiverem claramente identificados, o avaliador deve determinar "sobre que base" ou "com respeito a que" a avaliação deve ser feita. Isto significa que devem ser escolhidos os aspectos que devem ser enfatizados na avaliação [TREU94].

Todo projeto de interface do usuário deve ser avaliado, através de métodos que envolvam vários critérios de aceitação. Os critérios que podem ser utilizados são muitos, porém a partir de uma revisão da literatura [ABRA77, NEAL84, SHNE87, NAKA85, LEA88, DOWN92] procurou-se na tabela 3.1 sintetizar os critérios mais utilizados. As métricas de avaliação em uma interface gráfica podem ser associadas a uma janela de interação, a uma tarefa a ser executada, à duração de uma sessão interativa, ou ao produto como um todo.

| Critérios | Tarefa | Tarefa | Sessão | Produto como um todo |
|---|---|---|---|---|
| • Comandos disponíveis (relação de, e/ou número de): | ✓ | ✓ | ✓ | |
| • Comandos errados (relação de, e/ou número de) | ✓ | ✓ | ✓ | |
| • Comandos executados (relação de, e/ou número de) | ✓ | ✓ | ✓ | |
| • Combinação de cores presentes | ✓ | | | |
| • Comparação do uso de comandos executados com o número de comandos oferecidos | ✓ | ✓ | ✓ | |
| • Comparação entre dispositivos de entrada | | | | ✓ |
| • Discriminação das cores utilizadas | ✓ | | | |
| • Duração da | | | ✓ | |
| • Duração de uso de ajuda | ✓ | ✓ | ✓ | |
| • Duração na execução de uma tarefa | | | ✓ | |
| • Freqüência de apresentação de mensagens de Erros | ✓ | ✓ | ✓ | |
| • Freqüência de solicitação de ajuda on-line. | ✓ | ✓ | ✓ | |
| • Freqüência de tipos de erros. | ✓ | ✓ | ✓ | |
| • Freqüência de uso de vários comandos | ✓ | ✓ | ✓ | |
| • Freqüência de uso do sistema pelo usuário | | | | ✓ |
| • Solicitação de ajuda por | ✓ | ✓ | ✓ | |
| • Navegação nos menus (caminho) | | ✓ | ✓ | |
| • Nível de satisfação do usuário | | | | ✓ |
| • Número de tarefas completadas | | | ✓ | |
| • Número de vezes que um comando específico é usado. | ✓ | ✓ | ✓ | |
| • Objetos (discriminação dos, número de) | ✓ | | | |

**Tabela 3.1 Critérios de avaliação mais utilizados.**

| Critérios | Tarefa | Tarefa | Sessão | Produto como um todo |
|---|---|---|---|---|
| • Percurso na execução de uma tarefa | | ✓ | | |
| • Quantidade de ajuda solicitada | ✓ | ✓ | ✓ | |
| • Quantidade de Cores | ✓ | | | |
| • Quantidade de erros por | ✓ | ✓ | ✓ | |
| • Seqüência de comandos usados | ✓ | | ✓ | |
| • Sequenciamento | ✓ | | | |
| • Sequenciamento dos menus | | | | ✓ |
| • Tarefas abortadas (número de) | | | ✓ | |
| • Tarefas completadas (número de) | | | ✓ | |
| • Total de acertos | | | ✓ | |
| • Total de erros | | | ✓ | |

**Tabela 3.1 Critérios de avaliação mais utilizados(continuação)**

## *3.4 Fases do Ciclo de Vida do Produto em que Ocorre a Avaliação.*

Os métodos de projeto interativo permitem testar protótipos e revisá-los baseados em uma realimentação do usuário. Com o aumento na pesquisa e desenvolvimento das interfaces voltadas para as necessidades do usuário e, a evolução dos dispositivos de comunicação visual de alta resolução, nasceu uma nova abordagem de projeto de interfaces para sistemas computacionais - o projeto evolucionário, participativo ou sob influência do usuário. Os projetistas passaram a interagir com o usuário, durante a fase de projeto, no processo de desenvolvimento e por todo ciclo de vida do sistema [SHNE87]. Diversos pesquisadores acreditam que o desenvolvimento participativo traz contribuições positivas ao projeto[DEBR77, MORE88, HAWK91], pois permite facilmente testar os protótipos, revisá-los baseado em uma realimentação do usuário e implementar refinamentos sucessivos.

É de aceitação geral que a avaliação tenha lugar durante a fase de projeto, para assegurar que o desenvolvimento da interface não se afaste significativamente dos objetivos do projeto e das necessidades do usuário. Todavia, a fase em que será efetuada a avaliação depende da metodologia aplicada pelo projetista para o desenvolvimento da interface. [LEA88]. A Fig. 3.2. mostra o ciclo Projeto-Avaliação. Embora esta Figura sugira que a avaliação deva ser feita após a implementação, esta é apenas uma das possibilidades existentes.

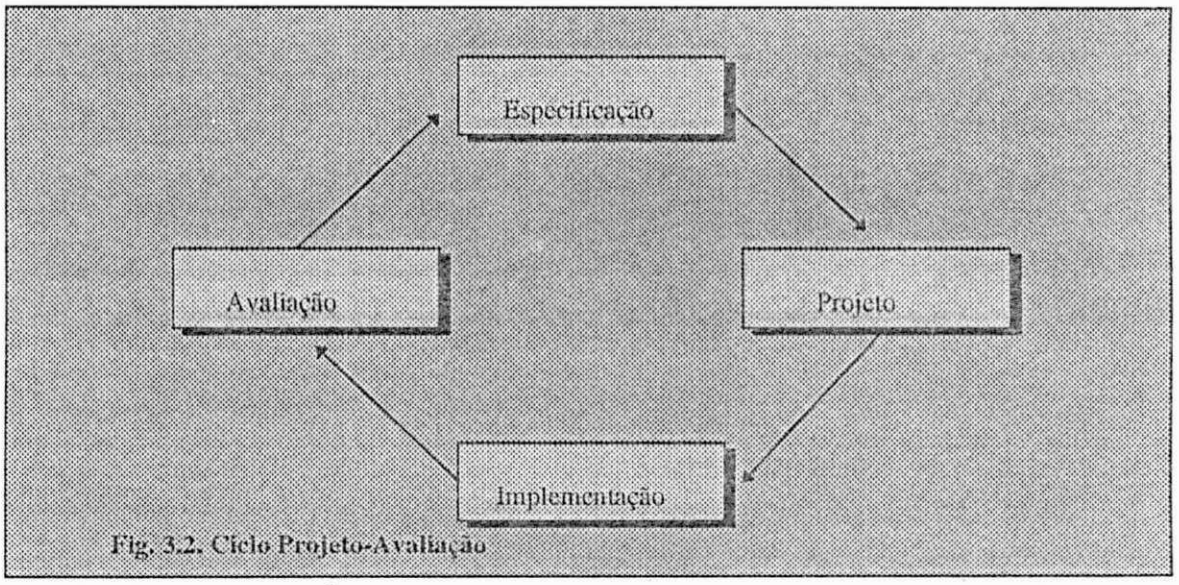


Fig. 3.2. Ciclo Projeto-Avaliação

Segundo Lea em [LEA88] os projetistas de interface podem fazer a avaliação através de simulações, permitindo que a avaliação seja feita antes da implementação. Deste modo as alterações no projeto podem ser feitas rapidamente, e evitando que os problemas existentes nesta fase sejam levados para uma fase posterior. A avaliação através de simulação consiste em apresentar ao usuário algumas características da interface, tais como: a estrutura dos comandos na forma de um manual, ou guia, onde o usuário é levado a realização de um conjunto de tarefas. A partir desta simulação é possível analisar as estratégias que poderão ser adotadas ou os comandos que ele usará, e os resultados que ele poderá obter do sistema.

A avaliação através de prototipadores é outra maneira de se identificar os problema de interfaces. Os prototipadores são como simuladores dinâmicos, que usam especialmente ferramentas de desenvolvimento de *software* para representar uma

interface mais complexa e, que tenha uma grande relação com o projeto. A Fig. 3.3 mostra como se processa a avaliação usando uma prototipação evolucionária.

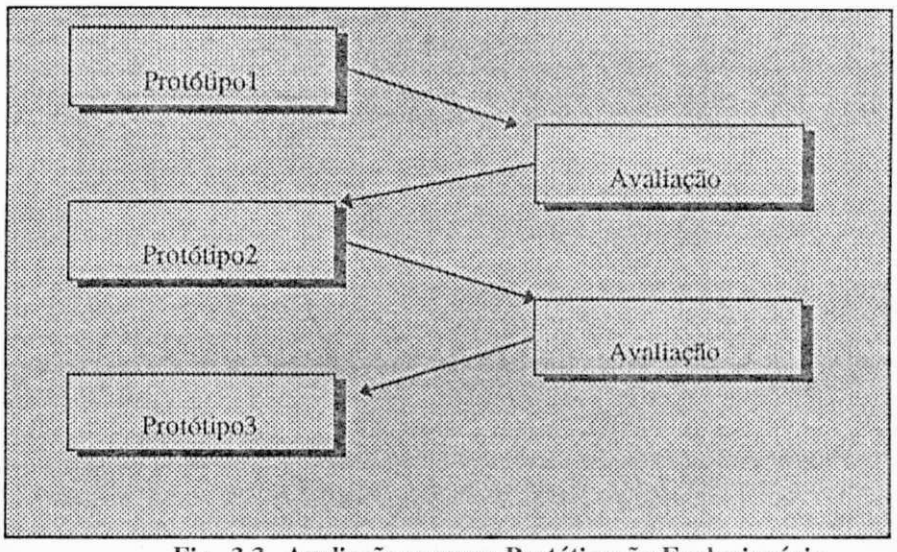



**Fig.. 3.3. Avaliação em uma Protótipação Evolucionária.**

A avaliação através de modelagem é o processo de desenvolvimento de um modelo formal da interface, ela é caracterizada pela modelagem de um número de procedimentos preestabelecidos. Primeiramente, uma tarefa é selecionada, de modo a ser representativa das tarefas que o sistema poderá realizar. A fase de análise da tarefa é seguida de um procedimento, no qual a ação requisitada pelo usuário para realizar a tarefa selecionada é apresentada. Este método é geralmente organizado dentro de um nível hierárquico de representação, começando no topo e procedendo a descida através dos níveis que compõem a hierarquia, até chegar ao nível de entrada/saída. Uma das vantagens da avaliação através do modelo conceitual da interface, é que ela pode ocorrer muito cedo dentro do processo de projeto, antes que maiores detalhes da especificação da interface tenham sido decididos. A avaliação da interface modelada pode ser baseada na descrição formal, ou utilizando métodos empíricos, dos quais falaremos posteriormente neste capítulo

## *3.5 Técnicas de Avaliação*

A avaliação de Interface HM, pode ser feita sob dois ângulos. Um deles se refere à analise das características ergonômicas da interface, que possibilita ao avaliador

observar se o projeto de desenvolvimento da interface está dentro dos padrões e diretrizes da área. O outro ângulo refere-se à análise das informações obtidas a partir da interação do usuário com a interface, permitindo que seja feita uma comparação do desempenho do usuário em relação às metas prefixadas ou em relação ao grupo a que ele pertence. Neste caso, se faz necessário que sejam coletadas todas as informações relevantes ao processo interativo e que sejam empregadas técnicas e métodos objetivos que viabilizem tal comparação.

Segundo Cox [COX93], a importância na escolha de técnicas e métodos decorre das necessidades impostas pela avaliação, que especifica quais medidas e comparações devem ser efetuadas face aos resultados esperados. Por conseguinte, como parte fundamental do planejamento da avaliação de uma determinada interface, o avaliador deve decidir quais técnicas e métodos devem ser mais apropriados. Uma vez que existem muitas técnicas de avaliação, que podem ser usadas individualmente ou em grupo, cabe ao avaliador saber escolhê-las dependendo da natureza da avaliação planejada, e decidir quando e como aplicá-las.

Segundo Treu em [TREU94], a metodologia da avaliação deverá possuir atributos análogos àqueles definidos pela metodologia de projeto; devendo ser:

- Sistemática - Apresentar uma clara organização do processo que leva à avaliação;
- Objetiva - Produzir resultados baseados em dados mensuráveis e confiáveis, (sempre que possível, uma vez que estes resultados se baseiam em dados subjetivos);
- Abrangente - Possibilitar a avaliação do composto Interface HM, a partir da avaliação seletiva de algumas partes ou de características de interesse;
- Explicativa- Mostrar o significado, método, fontes, critérios, razões, etc., que foram empregados para alcançar as conclusões da avaliação, a partir do exame dos registros realizados.

A utilização de uma metodologia de avaliação que forneça um diagnóstico o mais fiel possível dos problemas encontrados durante a avaliação, permitirá ao projetista, a

partir dos resultados, rever e corrigir o projeto, evitando assim que problemas venham a surgir quando da liberação da interface para o usuário final.

Segundo Matrin Lea em [LEA88], existe uma necessidade urgente na aprovação pelos projetistas e avaliadores, de um conjunto de critérios padronizados para avaliação de interfaces. Apesar da existência de diretrizes que poderiam ser um referencial com o qual os projetistas concordam, ainda se faz necessária uma discussão mais ampla para que estas diretrizes sejam de fato aceitas pela comunidade e possam ser utilizadas em todos os níveis de desenvolvimento da interface, a partir do nível conceitual.

Os métodos disponíveis para avaliação se classificam em dois grandes grupos - Formais (ou Analíticos) e Empíricos.

Ao operar uma interface, o usuário constrói um modelo conceitual sobre ela de modo a facilitar a interação. Os métodos Formais. - assumem que o modelo utilizado pelo projetista da interface pode ser uma aproximação razoável do modelo conceitual do usuário. Esta visão é justificada pela extensão com que o usuário pode facilmente assimilar o modelo do projetista a partir de: treinamento explícito, e/ou a partir da documentação, e/ou inferir sobre o modelo a partir do comportamento do sistema [TREU94]. O modelo que é utilizado em todos os níveis de desenvolvimento da interface, fornece uma descrição do sistema em seu próprio nível de abstração. Esta descrição consiste de procedimentos para realizar as tarefas em termos das ações disponíveis em cada nível. Este modelo é composto de vários componentes que atuam em três níveis. Conceitual; de Comunicação e Físico.

- O nível Conceitual contém os conceitos abstratos sobre os quais o sistema é organizado;
- O nível de Comunicação contém os comandos da linguagem e,
- O nível Físico trata dos dispositivos físicos com os quais usuário interage.

Os métodos Formais permitem que a partir da descrição da interface possam ser definidas algumas medidas de avaliação, tais como:

- Facilidade de aprendizagem;
- Eficiência do sistema e;
- Carga de memória.

A utilidade dos métodos Formais não dispensa a necessidade da avaliação através dos métodos Empíricos. Na verdade, o desenvolvimento e refinamento dos métodos formais, por si só, requer dados empíricos que permitam avaliar a metodologia formal. Os métodos Formais podem apenas fornecer uma representação incompleta da interação do usuário com a interface, enquanto que os métodos empíricos (simulação e protótipos) fornecem uma oportunidade de se avaliar o projeto usando os usuários.

Entre os métodos Empíricos existentes, Martin Lea em [LEA88] menciona:

- Métodos Cobertos e Descobertos
- Medidas Retrospectivas e Concorrentes
- Obtenção de dados Quantitativos e Qualitativos
- Obtenção de dados Subjetivos e Objetivos

A seguir passamos a descrever os métodos acima relacionados

- Os Métodos Cobertos são os métodos onde o usuário tem conhecimento da sua participação no processo avaliatório da interface, por exemplo a técnica de entrevista. Nos Métodos Descobertos, por outro lado, a coleta das informações que servirão para a avaliação da interface é feita de maneira discreta, sem que o usuário tenha conhecimento da sua participação no processo, por exemplo a técnica de monitoração dos dados.

- As Medidas Retrospectivas utilizam as informações, reconstruídas após a interação do usuário com a interface. As Medidas Concorrentes utilizam as informações no momento em que elas são geradas a partir da interação do usuário com a interface.

- A coleta dos Dados Quantitativos é um tipo de método que pode quantificar as observações registradas a partir de questionários, experimentos controlados, etc. A coleta dos Dados Qualitativos é um tipo de método que se baseia na opinião dos usuários sobre a interação e permitem aos avaliadores traçarem um perfil da qualidade da interface, a partir de anotações das observações, entrevistas desestruturadas, etc.

- Os Dados Subjetivos necessitam de interpretação a partir dos dados adquiridos através de questionários, escala de avaliação e da observação do processo de interação. Os dados Objetivos sobre o desempenho do usuário abrangem: o tempo gasto para completar uma tarefa, ou a freqüência de uso de diferentes comandos e apresentam um alto grau de precisão.

O métodos Formais e os Empíricos podem se utilizar de várias técnicas de avaliação, entre elas destacam-se:

- Questionários;
- Entrevistas;
- Observação da interação do usuário com o computador;
- Inspeção da interface ;
- Monitoração, Medição e Registro;
- Áudio ou Registro Visual;
- Análise da observação frente a modelos teóricos e,

- Questionários - especialmente projetados para a partir das respostas dos usuários fornecer subsídios aos analistas, para avaliar quantitativamente ou qualitativamente uma interface. As pessoas que estão sendo questionadas são responsáveis pela precisão das respostas, partindo do pressuposto que as perguntas foram bem planejadas de modo que não possuam ambigüidades.

  Os questionários podem ter duas versões:

  - Versão Convencional no papel
  - Versão em Tempo Real - *On-line*

- Entrevistas - representam uma forma mais dinâmica de se questionar os usuários, em substituição aos tradicionais questionários. Os resultados desta técnica dependem da habilidade do entrevistador e da abordagem utilizada. Se ele estruturou a entrevista baseado em uma estratégia bem planejada e formulou questões padronizadas para guiar a entrevista, será possível obter resultados diretos e confiáveis. Por outro lado, se ele conduziu a entrevista de forma desestruturada e usando um formato livre de abordagem sobre o assunto em questão, permitirá desta maneira que as informações resultantes não venham a fornecer dados corretos sobre a interação. Nesta técnica a responsabilidade sobre a qualidade dos dados é tanto do entrevistador quanto do entrevistado.
- Observação da interação do usuário com o computador - focaliza nas características observáveis e no desempenho do usuário durante a interação. Os dados resultantes refletem a ocorrência dos eventos durante a interação,

como também o julgamento efetuado sobre os mesmos. Os dados são registrados pelo observador, recaindo sobre ele a precisão dos resultados.

- Inspeção da interface - feita por um usuário experiente, focalizando nas características observáveis e no seu desempenho, visando a detecção de problemas específicos. As características dessa técnica são similares às da técnica *Observação da interação do usuário com o computador*, exceto que o observador está diretamente envolvido no processo.
- Monitoração, Medição e Registro - Esta técnica tem duas versões:
  - Engatilhada sob o controle do *software/hardware*
  - Direcionada por um observador humano que tem ajuda do *software*

Nesta técnica a exatidão dos dados depende da precisão do *software* e do *hardware* utilizados. A primeira versão requer um suporte adequado para monitoração baseada em computador. A segunda versão, envolve a estimulação pré-programada do usuário para reagir sempre que certas condições forem encontradas. A monitoração pode ser controlada e complementada por um experimentador *on-line* habilitado a observar (discretamente) o que o usuário está fazendo e, habilitado a introduzir mudanças tanto na funcionalidade quanto nos estímulos que são dados aos usuários em tempo real.

- Audio ou Registro Visual - registra o protocolo verbal (áudio) e ou o comportamento não verbal (vídeo). Tais dispositivos de registro podem ser usados como alternativa ou meio suplementar para coleta de dados efetuadas por outras técnicas. Nesta técnica a exatidão dos dados depende da clareza, ângulo de visão, etc. do elemento registrado. Naturalmente, os problemas aparecem na análise e interpretação de tais registros.
- Simulação - simulação do usuário, do computador, ou suas combinações. Esta técnica pode ser proveitosa para a obtenção de dados avaliatórios.
- Análise da observação frente a modelos teóricos - analisa o que foi observado, inspecionado, monitorado ou outra representação do desempenho da interface comparando com modelos teóricos. O analista é responsável pela exatidão dos resultados informados.

A escolha da técnica a ser aplicada na avaliação da interface vai depender da fase do ciclo de vida do produto em que o processo avaliatório vai ocorrer. Pode ocorrer na fase de especificação, implementação, antes da liberação do produto para o usuário ou mesmo depois do produto instalado, a escolha da técnica depende do contexto geral que vai ser avaliado, e/ou da disponibilidade dos usuários para participarem deste processo. Na maioria das vezes o projetista tem dificuldades em efetuar a avaliação devido a problemas relacionados com a impossibilidade de contar com a participação dos usuários que compõem o público alvo do produto, ou devido à falta de recursos para a avaliação, ou até mesmo por falta de tempo. Qualquer um desses motivos que impeçam a avaliação devem ser superados, pois todo esforço empregado poderá vir a ser recompensado com um produto de melhor qualidade.

# Capítulo IV

# Descrição da $FAI_{WIN}$

Apesar do aparecimento de um grande número de ferramentas de prototipagem de interfaces para o ambiente *Windows*, até onde pesquisamos, não foram encontradas na literatura ferramentas desenvolvidas para sua avaliação. Este capítulo apresenta o desenvolvimento de uma Ferramenta para Avaliação de Interfaces de aplicativos desenvolvidos para o Ambiente *Windows* - $FAI_{WIN}$.

## 4.1 Escopo da $FAI_{WIN}$

A $FAI_{WIN}$ é voltada para os projetistas e avaliadores de interfaces. Sua finalidade é fornecer dados coletados automaticamente ao longo de uma interação com o usuário, de modo a apoiar um processo de avaliação

Esta ferramenta fornece informações quantitativas sobre os elementos que compõem a interface e sobre a interação do usuário com a interface. Os avaliadores que a utilizarem estarão usado o método de avaliação empírico e a técnica de monitoração dos dados para a avaliação dos seus produtos.

Os dados fornecidos são equivalentes a uma transcrição da sessão e portanto, não refletem integralmente o tipo de informação disponível na leitura de transcrição da sessão. A $FAI_{WIN}$ necessita portanto, de uma técnica auxiliar do tipo: registro em vídeo, questionário, entrevistas, etc., que permita que alguns aspectos, que não são registrados por esta ferramenta, venham a complementar a avaliação, permitindo assim que o projetista possa fazer uma análise mais completa da sua interface.

## 4.2 Arquitetura da $FAI_{WIN}$

A $FAI_{WIN}$ é composta dos cinco (5) módulos relacionados a seguir:

1.Coletor de Dados Estáticos;

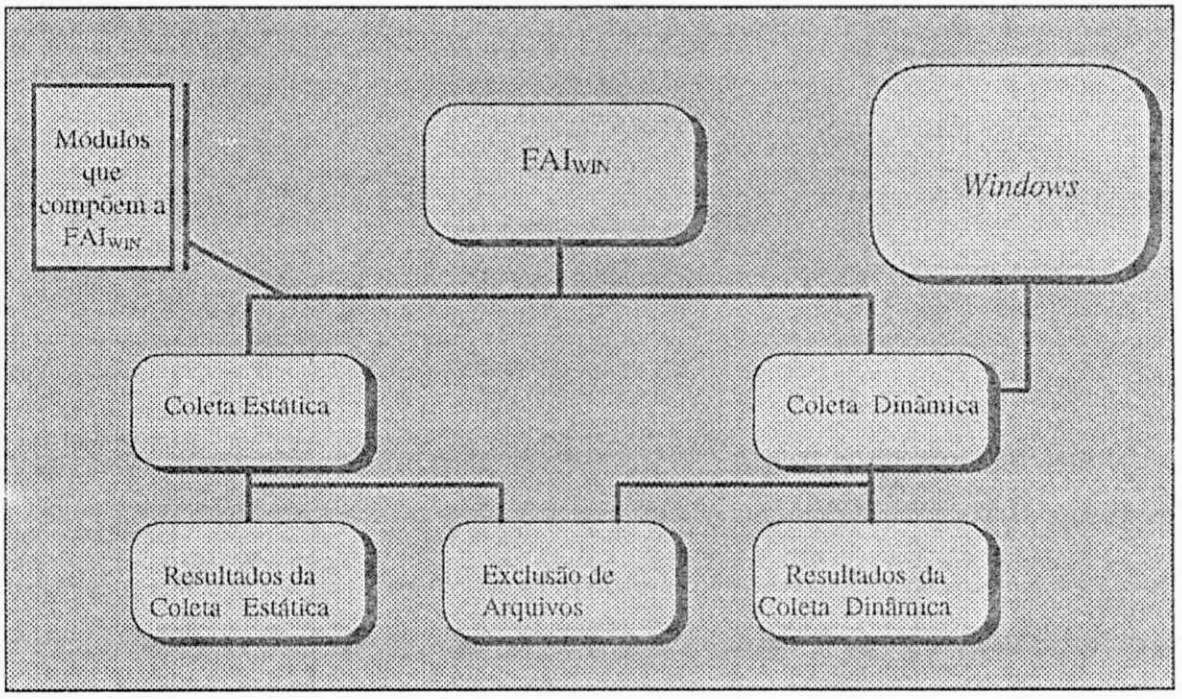
2.Coletor de Dados Dinâmicos;

3.Emissor dos Resultados da Coleta Estática;

4.Emissor dos Resultados da Coleta Dinâmica e

5.Um módulo de Exclusão de Arquivos.

A Fig. 4.1, apresenta graficamente a estrutura da $FAI_{WIN}$, incluindo a ligação da $FAI_{WIN}$ com o *Windows*, a partir do módulo da coleta dinâmica o qual mantém uma relação interna com os módulos de interrupção do *Windows*.




**. 4.1 Estrutura da $FAI_{WIN}$.**

## 4.3 Técnica utilizada pela $FAI_{WIN}$ para coleta de dados

Antes de detalhar a técnica utilizada pela $FAI_{WIN}$, passamos a conceituar os termos **tarefa** e **sessão** no contexto deste trabalho.

No que tange a interação HM, uma **sessão** corresponde a um período de tempo durante o qual o usuário interagiu com uma determinada interface. Uma **sessão** pode

consistir da execução de uma ou mais tarefas durante a realização de testes de desempenho.

Um elemento essencial do teste de usabilidade é a escolha das **tarefas** que o usuário deverá realizar com o produto. As metas que o teste deverá atingir determinam os tipos de **tarefas** a serem realizadas. Se o objetivo do teste de usabilidade for medir a eficácia de um programa de treinamento ou de um programa tutorial, a tarefa poderá consistir em treinar os usuários utilizando o próprio produto, incluindo os exercícios requisitados e, subsequentemente ao treinamento solicitar que realizem uma série de tarefas, visando aferir o aprendizado. Outras tarefas podem ser empregadas para avaliar a usabilidade e a produtividade deste produto. Estas tarefas deverão então ser representativas dos tipos de atividades que o usuário poderá realizar com o produto.

Como já foi mencionado, a técnica utilizada pela ferramenta aqui apresentada é a monitoração dos dados através do computador. Esta técnica consiste na captura automática de todas as entradas do usuário. Estes dados são coletados em tempo real e arquivados para uma posterior análise. Nesta técnica a ferramenta de coleta é empregada para registrar aspectos selecionados da sessão com o usuário. Os dados coletados durante uma sessão, contribuem significativamente para a avaliação da interação, pois são de baixo custo operacional e fornecem resultados em um menor tempo do que as técnicas que dependem do registro de interação em condições rotineiras de trabalho.

Segundo Siochi em [SIOCI91], a captura automática tem várias vantagens. Em primeiro lugar, os dados coletados representam as condições reais de interação do usuário com o sistema, ao contrário de outros tipos de coleta de dados em laboratório. Em segundo lugar, os dados armazenados em arquivos podem ser acessados por algoritmos de análise e técnicas de tratamento de dados. Neste caso as ferramentas analíticas liberam o avaliador do tédio associado com os outros tipos de análise, encorajando-o a fazer a avaliação. A coleta automática elimina também, a necessidade da presença de um observador, eliminando assim qualquer interferência sobre o usuário, ao mesmo tempo em que permite que seja efetuada a coleta de dados sobre vários usuários simultaneamente. Finalmente, as ferramentas analíticas habilitam uma análise rápida dos dados coletados fornecendo uma realimentação rápida para o projetista.

Uma ampla variedade de dados pode ser capturada por este método. Por exemplo, a ocorrência de certos eventos, tais como: ocorrência de erros ou de um

comando em particular, podem ser monitorados e suas freqüências de ocorrência calculadas em uma análise posterior.

Por outro lado, dentre as possíveis limitações dessa técnica pode-se destacar que, em geral, os dados coletados representam o contexto real de trabalho e não aquele de uma tarefa pré-definida. Desta forma é difícil identificar, o que o usuário estava tentando fazer em algum instante da coleta. Este problema, no entanto, pode ser resolvido se após a coleta, em entrevista com o usuário o projetista vir a esclarecer questões desta natureza. Uma outra desvantagem aparente é que o usuário tende a usar um subconjunto de recursos da aplicação. Todavia essa limitação passa a fornecer um elemento chave para posterior análise, quando o projetista poderá observar o que não está sendo utilizado, levado-o a rever o projeto. Nesta revisão verifica-se a existência de possíveis falhas na elaboração da interface, ou a necessidade de um treinamento mais aprofundado.

Finalmente uma restrição desse método é que dados capturados são frios e impessoais, deixando de apresentar algumas informações necessárias relacionadas com a realização das tarefas do usuário. Porém, qualquer técnica que capture este aspecto poderá vir a complementar o sistema de monitoração.

Dentre os diversos pontos positivos desse método é possível destacar o alto grau de confiabilidade dos dados coletados. Estes dados podem formar a base para comparações objetivas entre interfaces e sobre variáveis de desempenho, uma vez que os dados são obtidos "discretamente" sem interferir sobre o desempenho do usuário.

## *4.4 Interface da FAI$_{WIN}$.*

A FAI$_{WIN}$ contém uma interface gráfica com o usuário, onde o projetista poderá selecionar os aspectos a serem analisados. A interface da ferramenta, bem como a interface dos módulos que a compõem, foram prototipadas utilizando a ferramenta de prototipagem do *Visual Basic*. A interface da FAI$_{WIN}$ é composta das seguintes janelas:

- ✓ Janela Principal (Fig. 4.2);
- ✓ Janela de Finalização do Processamento (Fig. 4.3)
- ✓ Janela de Informações sobre a Ferramenta (Fig.4.4)

- ✓ Janela de Entrada de Dados do Projeto (Fig. 4.7).
- ✓ Janela de Informações sobre a Abertura do Arquivo(Fig. 4.8).
- ✓ Janela de Abertura de Arquivo (Fig. 4.9).
- ✓ Janela de Confirmação do Processamento (Fig. 4.10).
- ✓ Janela de Cancelamento do Processamento (Fig. 4.11).
- ✓ Janela de Informações sobre o Processamento (Fig. 4.12).
- ✓ Janela de Informações sobre o Usuário e a Sessão de Coleta (Fig. 4.15).
- ✓ Janela de Confirmação da Coleta Dinâmica (Fig. 4.16)
- ✓ Janela de Informações sobre a Ocorrência de Erros na Coleta (Fig. 4.17).
- ✓ Janela de Informações sobre a Versão da Coleta (Fig. 4.18)
- ✓ Janela de Resultados da Coleta Estática (Fig. 4.19).
- ✓ Janela de Informações sobre os Objetos em uma Janela (4.20).
- ✓ Janela de Emissão das Cores em uma Janela (Fig. 4.21).
- ✓ Janela de Informações sobre os Comandos em uma Janela (Fig. 4.22).
- ✓ Janela de Informações sobre o Usuário, a Versão e Sessão da Coleta (Fig. 4.23).
- ✓ Janela de Resultados da Coleta Dinâmica (Fig. 4.24)
- ✓ Janela com Informações sobre uma Sessão (Fig. 4.25).
- ✓ Janela com Informações sobre uma Tarefa (Fig. 4.26).
- ✓ Janela com Informações sobre uma Janela (Fig. 4.27).
- ✓ Janela para Exclusão de Arquivos (Fig. 4.28 ).
- ✓ Janela de Confirmação da Exclusão de Arquivo (Fig. 4.29);

A seguir serão descritas algumas das janelas da $FAI_{WIN}$, que compõem a ferramenta. As janelas relativas às opções da $FAI_{WIN}$ serão detalhadas quando forem descritos os módulos correspondentes.

❑ Janela Principal - esta janela contém um menu, onde o projetista poderá selecionar os módulos a serem executados. Entre os itens a serem selecionados encontram-se os itens sair e ajuda. O item sair permite que seja finalizado o processamento da $FAI_{WIN}$ e o item ajuda causa a abertura da Janela de Informações sobre a Ferramenta.

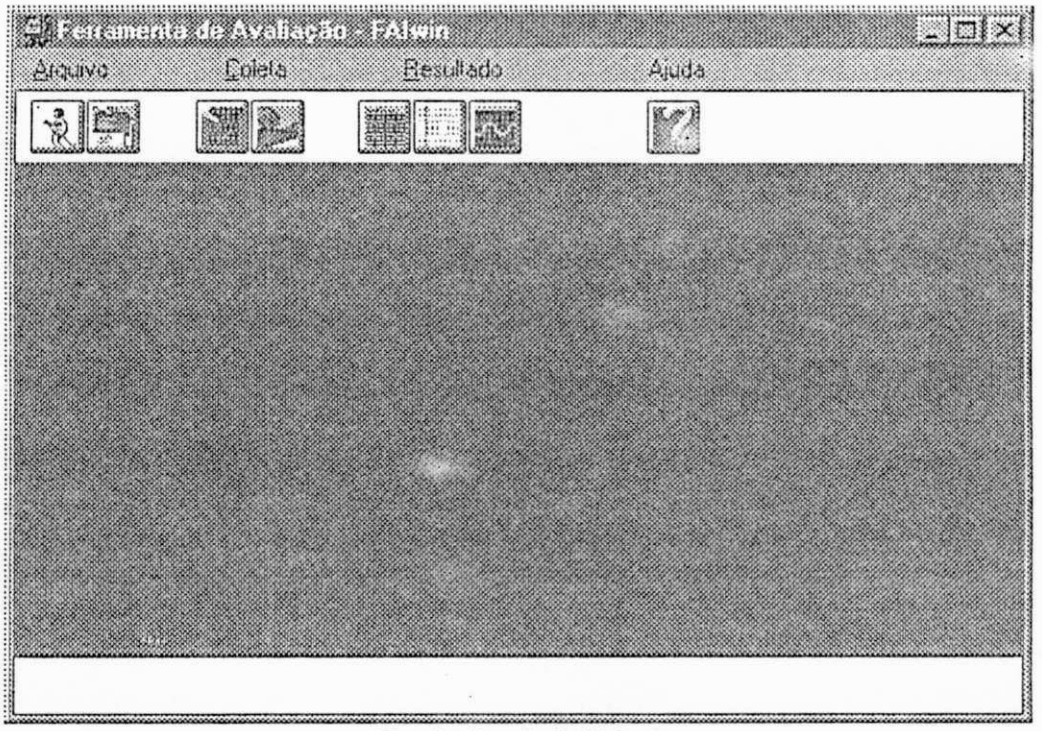



**Fig. 4.2 Janela Principal**

❑ Janela de Finalização do Processamento - nesta janela o usuário poderá optar por encerrar o processamento, ou voltar à Janela Principal (caso desista de finalizar o processamento) pressionando o botão cancela.

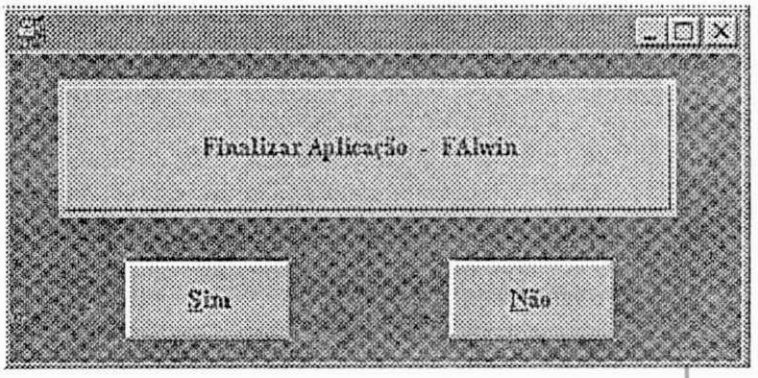



**Fig 4.3 Janela de Finalização do Processamento**

❑ Janela de Informação sobre a Ferramenta - esta janela contém informações sobre a autoria da ferramenta.

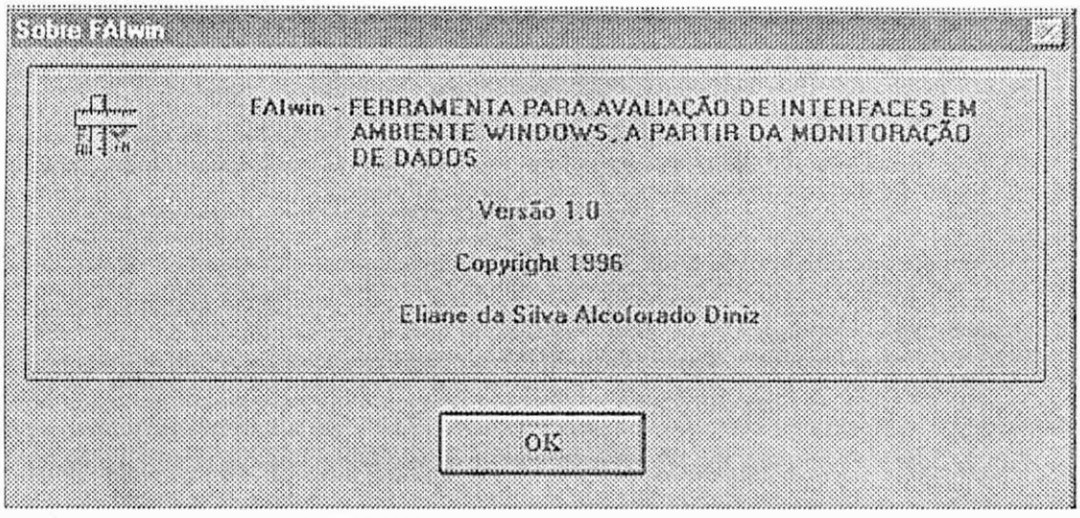



**Fig. 4.4 Janela de Informações sobre a Ferramenta**

A FAI$_{WIN}$ possui um sistema de ajuda *on-line* em todos os seus módulos. Nos módulos referentes a apresentação dos resultados, além da ajuda, também é disponibilizado um botão de opção, que abre uma janela, a qual contém uma relação de diretrizes de projeto de interface, referente ao resultado selecionado.

Na versão atual da FAI$_{WIN}$ a apresentação dos resultados da avaliação, na forma gráfica (gráfico de pizza, barras, etc.) é feita fora da ferramenta, foi utilizada a planilha *Excel,* porém em versões futuras está prevista a inclusão de um módulo para geração destes gráficos dentro do ambiente da ferramenta FAI$_{WIN}$.

## *4.5 Descrição Funcional.*

A FAI$_{WIN}$ permite que o avaliador realize uma avaliação quantitativa, observando os aspectos estáticos (avaliação estática) e / ou dinâmicos (avaliação dinâmica) da interface. Entende-se por avaliação estática a análise das informações contidas na definição da interface, feita em tempo de projeto. Estas informações independem da interação com o usuário e consistem de: relação das cores presentes nas janelas; quantidade total de objetos em cada janela; a quantidade total de cores em cada janela, etc. Por outro lado entende-se por avaliação dinâmica, a análise dos dados obtidos a

partir da interação de usuários com o produto. A título de exemplo pode-se citar como informações dinâmicas: taxa de utilização de comandos, taxa de execução de tarefas, taxa de solicitação de ajuda; etc.

A seguir será descrita a funcionalidade dos módulos que compõem a $FAI_{WIN}$.

### *4.5.1 Coletor Estático (CE)*

Como foi visto na seção 4.5, a Avaliação Estática de uma interface aborda os aspectos da interface do ponto de vista das decisões de projeto. Para a obtenção dos dados que permitirão esta avaliação, se fez necessário a implementação de um Coletor Estático. Os dados coletados servirão para que os projetistas possam verificar se o seu projeto está dentro das diretrizes e normas de especificação de Interfaces. Dentre as características investigadas pela $FAI_{WIN}$ pode-se destacar as propriedades das janelas que compõem a interface, tais como: cor de fundo, cor de frente, cor das bordas, estilo de preenchimento do janela, espessura da bordas, etc. A $FAI_{WIN}$ também coleta informações sobre cor e outras propriedades relativas aos objetos que compõem cada janela. A seguir serão relacionados os arquivos utilizados como entrada de dados pelo CE.

#### *4.5.1.1 Arquivos utilizados como entrada de dados pelo CE*

Inicialmente o arquivo de Projeto (.MAK) gerado pela ferramenta de prototipagem, neste caso *Visual Basic*(VB), é utilizado como entrada de dados. Este arquivo contém uma lista de todos os arquivos necessários para a construção da interface que está sendo projetada:

- ❑ Arquivos de formulários (.FRM). Um arquivo por janela;
- ❑ Arquivos de modulo de código (.BAS);
- ❑ Arquivos de extensão do VB (.VBX). Estes arquivos contém informações sobre os controles personalizados, existentes na caixa de ferramenta do VB.

Em seguida, o coletor estático seleciona os arquivos texto (.FRM) gerados pelo prototipador da Interface. Estes arquivos contém a descrição gráfica dos formulários (Janelas) e dos seus controles, incluindo as suas propriedades. Estas propriedades descrevem a aparência e o comportamento da interface que será analisada. Os arquivos

.FRM, são arquivos gerados em ASCII. Para sua geração escolhe-se na janela de projeto do VB o menu ARQUIVO, a seguir o item SALVAR COMO na caixa de diálogo correspondente e, assinala-se a opção SALVAR COMO TEXTO.

### *4.5.1.2 Estruturas de Dados utilizadas pelo CE*

O CE utilizou como estruturas internas, duas listas lineares simplesmente encadeadas. Na primeira lista (Fig. 4.5.), cujos campos estão detalhados na Tab.4.1., cada nodo contém o nome do arquivo de formulários (.FRM) encontrado. Ao completar a leitura do arquivo de Projeto, inicia-se o preenchimento da segunda lista (Fig. 4.6.), cujos campos estão detalhados na Tab. 4.2. Nesta lista cada nodo contém um objeto pertencente ao respectivo formulário. Ao iniciar um novo formulário a lista previamente preenchida é armazenada em disco e o seu conteúdo "desalocado" para permitir a geração de uma nova lista.

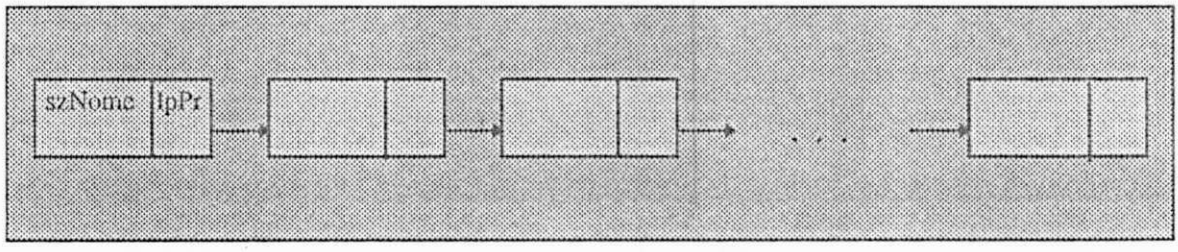



**Fig. 4.5. Estrutura do Coletor Estático com os nomes dos formulários.**

| Campos da Estrutura | | Descrição |
|---|---|---|
| Tipo | Nome | |
| char | szNome[81] | Nome do Formulário |
| Struct | lpPr | Ponteiro para o próximo nodo da lista |

**Tab. 4.1 Campos que compõem cada nodo da Fig. 4.5.**

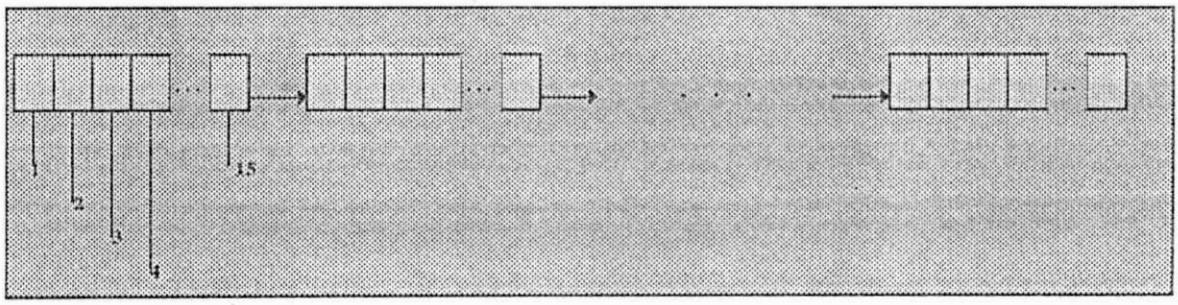



**Fig. 4.6 Estrutura do Coletor Estático com os elementos selecionados no arquivo de formulário.**

| Campos da Estrutura | | Função |
|---|---|---|
| **Tipo** | **Nome** | |
| char | (1) szNomeObj[41] | Nome do Objeto |
| char | (2) szTipoObj[41] | Tipo do Objeto |
| char | (3) szTipoObj[41] | Título do objeto |
| long | (4) lCorTitulo | Cor do Título |
| long | (5) nEstiloBarra | Estilo da Barra |
| long | (6) lCorBarra | Cor da Barra |
| long | (7) nEstiloFundo | Estilo da cor de fundo |
| long | (8) lCorFundo | Cor de fundo |
| long | (9) nEstiloFrente | Estilo da cor de frente |
| long | (10) lCorFrente | Cor de frente |
| long | (11) nEstiloBorda | Estilo da Borda |
| long | (12) lCorBorda | Cor da Borda |
| long | (13) nEstiloPreencher | Estilo de preenchimento dos objetos |
| long | (14) lCorPreencher | Indica a cor de preenchimento de determinados objetos ex.: Shapes |
| struct | (15) lpProxObj | Ponteiro para o próx. nodo da lista |

**Tab. 4.2 Descreve os campos que compõem cada nodo da Fig. 4.6.**

### *4.5.1.3. Arquivos gerados pelo CE*

No CE foram gerados dois arquivos, utilizando o modo binário. No primeiro arquivo (Arquivo de dados) foram armazenados os dados da coleta de acordo com a estrutura de dados (Fig. 4.3), eliminando-se o apontador da lista. Estes dados serão utilizados durante a fase de análise do protótipo que está sendo avaliado. O nome do arquivo gerado pelo CE é formado por:

**SiglaVersao.est**

Onde, **Sigla** é composta pelos primeiros seis (6) caracteres da sigla do projeto, e é fornecida pelo usuário.

**Versão** é um número de dois dígitos fornecido pelo usuário, e corresponde à versão do projeto que está sendo avaliado.

**est** é a extensão do nome do arquivo, indicando que se trata de dados de coleta estática.

**est** é a extensão do nome do arquivo, indicando que se trata de dados de coleta estática.

No segundo arquivo foram armazenadas informações sobre o Projeto (Tab.4.3.) e o nome do arquivo da coleta estática efetuada. A finalidade deste arquivo é fornecer subsídios ao projetista para fazer uma avaliação entre as várias versões do protótipo que está sendo analisado. O nome do arquivo tem a seguinte regra de formação:

**Sigla.pje**

Onde, **Sigla** corresponde à sigla fornecida pelo usuário. Este *string* não pode exceder 8 caracteres.

**pje** é a terminação escolhida para indicar projeto estático.

| Campos da Estrutura | | Descrição |
|---|---|---|
| **Tipo** | **Nome** | |
| char | szNomeSist[60] | Nome do Sistema |
| char | szNomeProj[30] | Nome do Projeto |
| char | szNomeProjt[30] | Nome do Projetista |
| char | szSigla[9] | Sigla do Projeto |
| char | szVersao[3] | Determina a versão do Protótipo |
| char | szDataColeta[8] | Data da coleta |
| char | szNomeArq[13] | Nome do arq. da coleta estática |

**Tab. 4.3. Descreve os campos que compõem os registro do arquivo de projeto da coleta estática.**

### *4.5.1.4 Interface com o usuário do CE.*

O módulo do Coletor Estático possui uma interface com o usuário no estilo *Windows*.

A interface é composta de:

- ✓ Janela de Entrada de Dados do Projeto (Fig. 4.7);
- ✓ Janela de Informações sobre a Abertura do Arquivo (Fig. 4.8).
- ✓ Janela de Abertura de Arquivo (Fig. 4.9);
- ✓ Janela de Confirmação do Processamento (Fig. 4.10.);
- ✓ Janela de Cancelamento do Processamento (Fig. 4.11);
- ✓ Janela de Informações sobre o Processamento (Fig. 4.12).

A seguir será feita uma breve descrição de cada janela.

❑ Janela de Entrada de Dados do Projeto - é a janela que possibilita o cadastramento do projeto da interface a ser avaliada. Tais como:

- ☒ Sigla do projeto;
- ☒ Versão do projeto;
- ☒ Nome do sistema;
- ☒ Nome do projeto;
- ☒ Nome do projetista;
- ☒ Data da coleta.

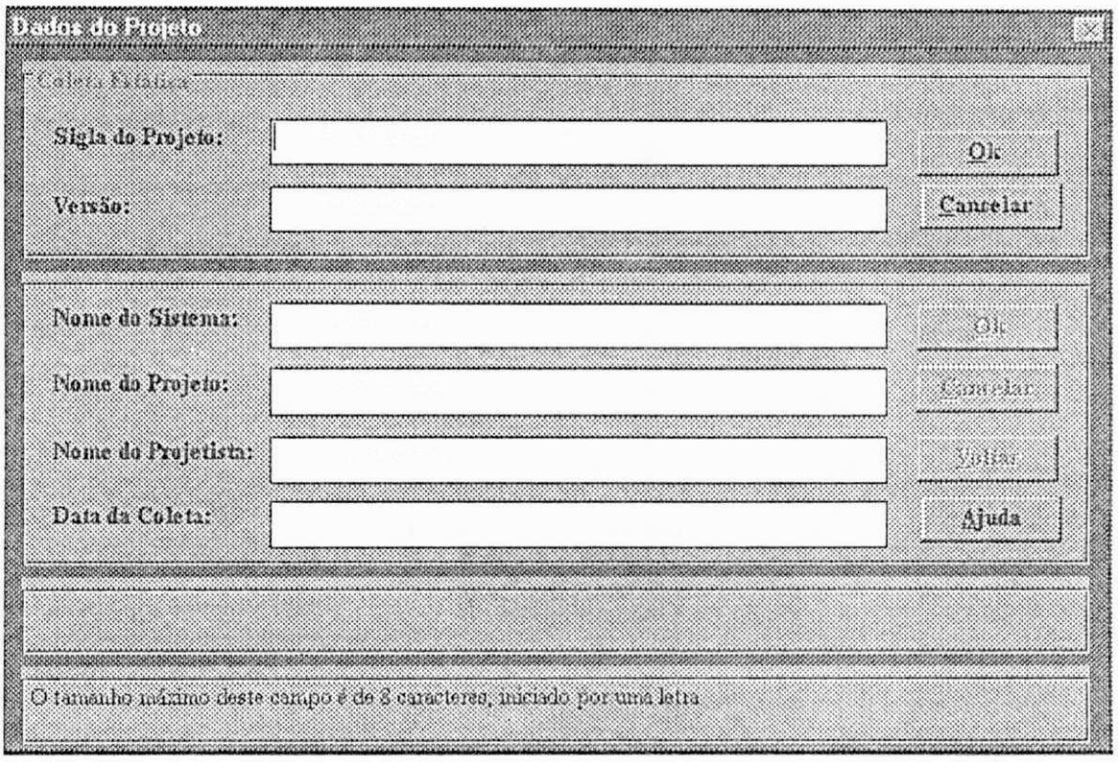



**Fig. 4.7 Janela de Entrada de Dados do Projeto**

❑ Janela de Informações sobre a Abertura do Arquivo - é a janela que apresenta o resultado da operação de abertura do arquivo. Se a operação foi bem sucedida, será ativada a janela de Confirmação do Processamento, caso contrário a Janela de Abertura de Arquivo ficará ativa esperando uma intervenção do usuário.

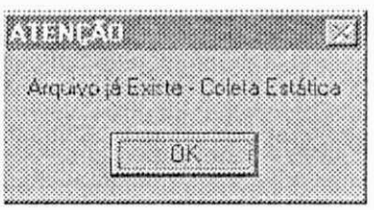



**Fig 4.8 Janela de Informações sobre a Abertura do Arquivo**

❑ Janela de Abertura de Arquivo - é uma caixa de diálogo, que permite que o usuário selecione o nome do arquivo de projeto, cuja extensão é .MAK. Este arquivo é criado pelo *Visual Basic* quando da prototipação da interface.

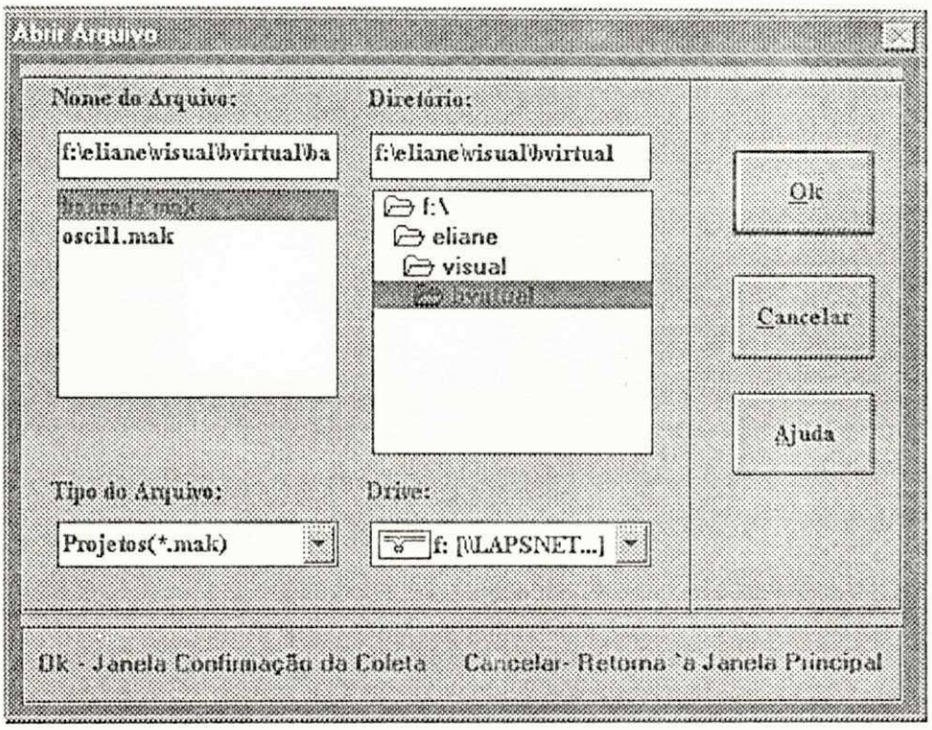



**Fig 4.9 Janela de Abertura de Arquivo**

❑ Janela de Confirmação do Processamento - uma vez programada a coleta a partir dos dados de identificação, o analista pode dar prosseguimento ao processamento da coleta ou retornar à Janela de Abertura de Arquivo, para nova programação.

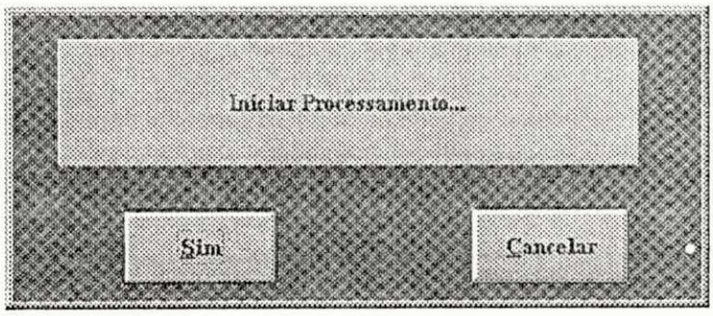



**Fig. 4.10 Janela de Confirmação do Processamento**

❑ Janela de Cancelamento do Processamento - uma vez iniciada a coleta de dados, esta janela permite que a qualquer momento o analista interrompa o processamento de coleta.

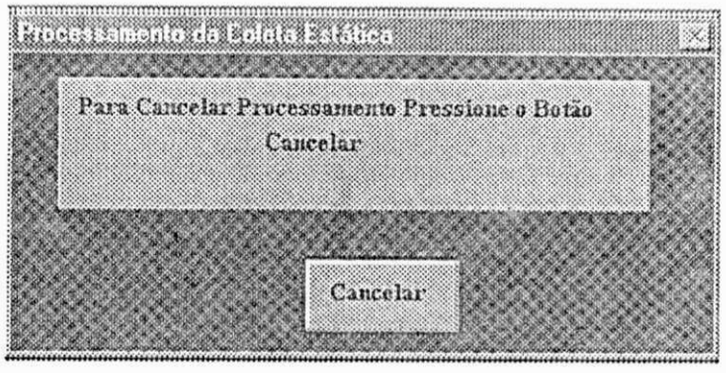



**Fig. 4.11 Janela de Cancelamento do Processamento**

❑ Janela de Informações sobre o Processamento - relata as ocorrências do processamento.

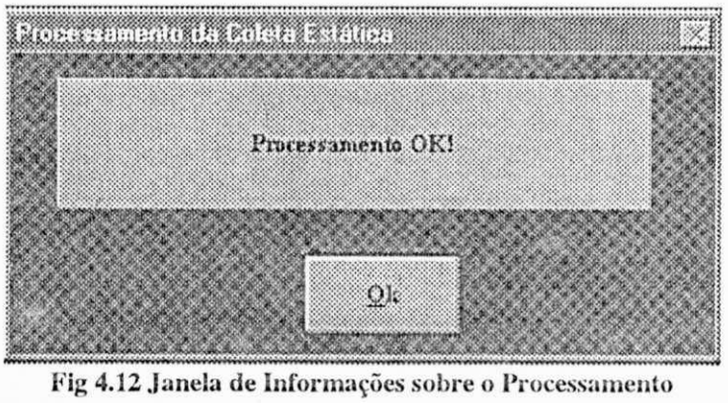



**Fig 4.12 Janela de Informações sobre o Processamento**

### *4.5.2 Coletor Dinâmico (CD)*

A coleta dinâmica é um tipo de coleta que obtém os dados de forma automática, a partir da interação de usuários com a interface. Uma ampla variedade de dados podem ser capturados durante uma sessão com o usuário, entre outros: a tecla que foi pressionada, o acionamento do *mouse*, etc. O dado coletado automaticamente é confiável, e uma grande quantidade de dados pode ser registrada em um curto intervalo de tempo.Este tipo de coleta é freqüentemente utilizado na avaliação de sistemas *on-line*, onde as tarefas têm um elevado grau de precisão. Nestes casos os dados da interação são obtidos sem interrupção e sem afetar o desempenho do sistema.

Para efetuar esta coleta foi desenvolvido um coletor dinâmico, o qual será descrito a seguir.

#### *4.5.2.1 Entrada de Dados do CD*

A entrada de dados no coletor dinâmico resulta da interação do usuário com a interface que está sendo avaliada. Devido ao grande volume de dados gerados em uma sessão de interação é necessário selecionar aqueles de interesse para a análise, evitando assim coletas desnecessárias. Os dados coletados pela $FAI_{WIN}$ são apresentados na sessão 4.5.3.

#### *4.5.2.2 Estruturas de Dados utilizadas pelo CD*

O **CD** utiliza como estruturas internas, dois vetores. A opção pelo uso de vetores decorreu da necessidade de agilizar o processo de coleta. A monitoração dos dados durante a coleta dinâmica, exige a utilização de um mecanismo interno do *Windows* que intercepta os eventos antes que eles cheguem a aplicação a que se destinam - o *hook*. Durante esta fase a $FAI_{WIN}$ fica em *background*, isto é não possuí o foco de entrada. O *hook* fornece à aplicação um meio de poder laçar um evento que esteja ocorrendo em uma dada aplicação, ou no sistema *Windows* [RICH92]. Um *hook* é uma subrotina instalada no tratamento de mensagens do *Windows*. Logo, ele permite que uma aplicação possa monitorar certos tipo de eventos gerados no ambiente *Windows* [YAO95].

Na $FAI_{WIN}$ os eventos selecionados pelo avaliador serão capturados e armazenados nos vetores correspondentes. Uma permanência mais prolongada nos *hooks* poderia afetar o desempenho do próprio *Windows*, daí a opção por estruturas do tipo vetor, os quais são criados estaticamente ao contrário das listas de criação dinâmica.

A $FAI_{WIN}$ utilizou três tipos de *hooks*:

- O *hook* de teclado (*WH_KEYBOARD*) para captura de teclas;
- O *hook WH_CBT* que controla as ações sobre a janela, (criação, redimensionamento, ativação da janela, etc.) e
- O *hook WH_SYSMSGFILTER* que controla as mensagens processadas por uma caixa de diálogo, um menu ou uma barra de rolamento.

pelo usuário (apenas aquelas selecionadas pelo avaliador). Ao preencher os vetores, seus conteúdos serão descarregados em um arquivo em disco para que possam armazenar novos eventos.

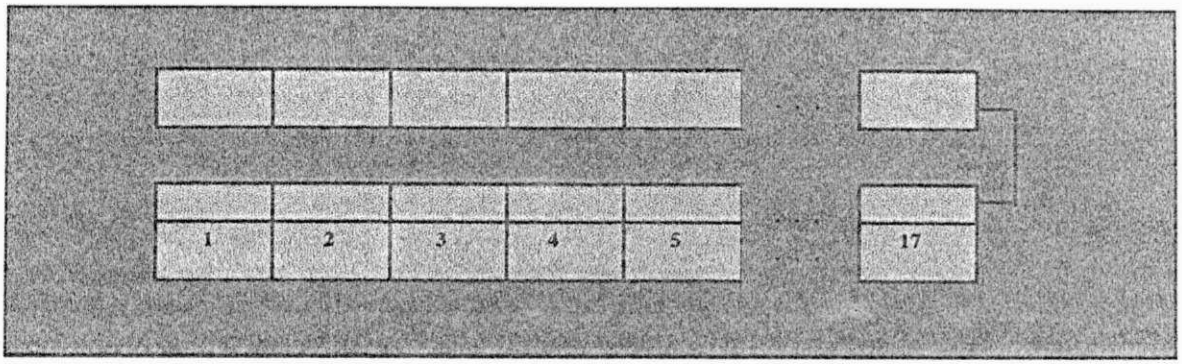



**Fig. 4.13 Estrutura de veto res do Coletor Dinâmico com os dados da janela ativa.**

| Campos da Estrutura | | Descrição |
|---|---|---|
| **Tipo** | **Nome** | |
| int | (1) CodTecla | Cód. da última tecla, antes da ativação da janela |
| int | (2) MEvento | Informa se a janela foi ativada pelo *mouse* |
| int | (3) CodJanAnt | Ident. janela anteriormente ativada |
| int | (4) CodJan | Ident. janela que está sendo ativada |
| char | (5) NomeJan | Nome da classe da janela |
| int | (6) Ti_hora | Hora da ativação da janela |
| int | (7) Ti_min | Minuto da ativação da janela |
| int | (8) Ti_seg | Segundo da ativação da janela |
| int | (9) Ti_hund | Centésimo da ativação da janela |
| int | (10) Tf_hora | Hora da desativação da janela |
| int | (11) Tf_min | Minuto da desativação da janela |
| int | (12) Tf_seg | Segundo da desativação da janela |
| int | (13) Tf_hund | Centésimo da desativação da janela |
| char | (14) TemMenu | Informa se a janela possui Menu |
| short | (15) AtivouItem | Informa qual dispositivo ativou o Menu -teclado ou mouse |
| char | (16) RotItem[41] | Rotulo do item selecionado |
| long | (17) TM_Hora | Hora da ativação do item |

Tab 4.4. Campos que compõem cada vetor da Fig. 4.13.

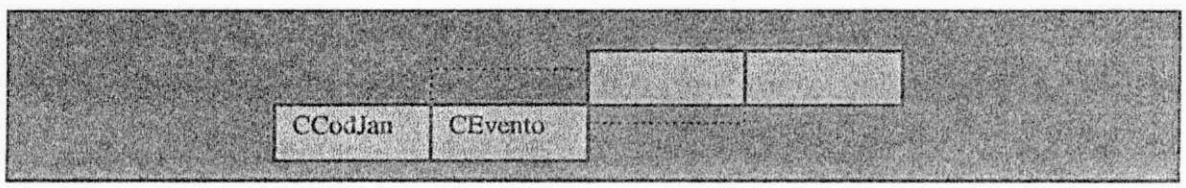



**Fig. 4.14. Estrutura do Coletor Dinâmico com os códigos das teclas pressionadas.**

| Campos da Estrutura | | Descrição |
|---|---|---|
| **Tipo** | **Nome** | |
| int | CCodJan | Código da janela Ativa |
| int | CEvento | Código da tecla pressionada. |

**Tab. 4.5. Campos que compõem cada nodo da Fig. 4.14.**

### *4.5.2.3. Arquivos gerados pelo CD*

No CD foram gerados três arquivos, utilizando o modo binário. No primeiro arquivo (arquivo de informações sobre janela) foram armazenados os dados da coleta de acordo com a estrutura de dados (Fig. 4.13). O nome do arquivo gerado pelo CD tem a seguinte regra de formação:

**SiglaVersaoNumSessaoNumUsuario.din**

Onde, **Sigla** é composta pelos primeiros dois (2) caracteres da sigla do projeto fornecida pelo usuário.

**Versão, NumSessao e NumUsuario** - cada elemento é formado por um número de dois dígitos fornecido pelo usuário.

**din** é a extensão do nome do arquivo de dados do tipo dinâmico.

No segundo foram armazenados os dados da coleta, referente as teclas pressionadas em uma determinada janela e o identificador desta janela, de acordo com a estrutura de dados (Fig. 4.14.). O nome do arquivo gerado pelo CD tem a seguinte regra de formação:

**SiglaVersaoNumSessaoNumUsuario.eve**

Onde, **Sigla** é composta pelos primeiros dois (2) caracteres da sigla do projeto fornecida pelo usuário.

**Versão, NumSessao e NumUsuario** - cada elemento é formado por um número de dois dígitos fornecido pelo usuário.

**eve** identifica o arquivo de eventos da coleta dinâmica.

No terceiro arquivo (arquivo de projeto) foram armazenadas informações sobre o Projeto (Tab.4.6.), o nome do arquivo da coleta estática efetuada e o nome do arquivo de eventos. A finalidade deste arquivo é fornecer subsídios ao projetista para fazer uma avaliação entre as várias versões do protótipo que está sendo analisado. O nome do arquivo tem a seguinte nomenclatura:

**Sigla.pjd**

Onde, **Sigla** corresponde a Sigla fornecida pelo usuário, com no máximo 8 caracteres.

**pjd** é a terminação escolhida para identificar o projeto dinâmico.

| Campos da Estrutura | | Descrição |
|---|---|---|
| **Tipo** | **Nome** | |
| char | szNomeSist[61] | Nome do Sistema |
| char | szNomeProj[31] | Nome do Projeto |
| char | szNomeProjt[31] | Nome do Projetista |
| char | szSigla[9] | Sigla do Projeto |
| char | szVersao[3] | Determina a versão do Protótipo |
| char | szDataColeta[8] | Data da coleta |
| int | nTipoUsuario | Tipo do usuário |
| char | szNunUsuario | Número do usuário |
| char | szSessao[3] | Identificação da Sessão |
| char | szNomeArq1[13] | Nome do arq. da coleta Dinâmica |
| char | szNomeArq2[13] | Nome do arquivo de eventos |

**Tab. 4.6. Campos que compõem os registro do arquivo de projeto da coleta estática.**

*4.5.2.4 Interface com o usuário do CD.*

O módulo do Coletor Dinâmico possui uma interface que é composta das seguintes janelas:

- ✓ Janela de Entrada de dados do Projeto (Fig. 4.9.);
- ✓ Janela de Informações sobre o Usuário e a Sessão de Coleta (Fig. 4.15.)
- ✓ Janela de Informação sobre a Abertura do Arquivo (Fig. 4.8);
- ✓ Janela de Abertura de Arquivo (Fig. 4.9.);
- ✓ Janela de Confirmação da Coleta Dinâmica (Fig. 4.16.);
- ✓ Janela de Informação sobre a Ocorrência de Erro na Coleta (Fig. 4.17.).

A seguir será feita uma breve descrição de cada janela.

❑ Janela de Entrada de Dados do Projeto - é a janela que possibilita o cadastramento do projeto a partir do fornecimento das informações: nome do sistema, sigla do projeto, versão atual do projeto a ser avaliado, etc. (Seção 4.5.1.4)

❑ Janela de Informações Adicionais - esta janela permite que o projetista forneça informações sobre:

☒ O número do usuário - este número identifica o usuário e segue uma seqüência pré-definida pelo projetista;

☒ O número da sessão, identifica a sessão.

☒ O perfil do usuário;

A população alvo para qualquer produto que esteja sendo construído, deve ser bem determinada. Na maioria das vezes, a especificação do produto é vaga em relação ao seu público alvo. Para ser aplicado o teste de usabilidade de qualquer produto, se faz necessário que, uma quantidade representativa desses usuários seja selecionada e o seu perfil sejam definidos. Na atual versão da $FAI_{WIN}$, foram consideradas 3 classes de usuários que vão interagir durante o processo de avaliação de uma interface. A

classificação se deu com base no seu conhecimento prévio da tarefa e na freqüência esperada para uso do sistema.

Há várias formas de classificar os usuários. Ao se reconhecer a diversidade de seu comportamentos, objetivos e perfis, este conhecimento pode levar a diferentes objetivos de projeto.

Uma separação genérica, é aquela feita por Shneiderman em [SCHNE 92]. Lá os usuários são classificados em: noviços ou iniciantes, experientes com uso ocasional do sistema e, em especialistas com uso freqüente.

A primeira classe - usuários noviços ou iniciantes, apresenta pouco conhecimento sintático sobre o sistema e pouco conhecimento semântico sobre o uso dos computadores, e apresentam pouco conhecimento sobre a tarefa.

A classe dos usuários ocasionais e experientes, mantém o conhecimento semântico sobre a tarefa e os conceitos relativos ao computador, mas retem com dificuldade o conhecimento sintático.

Finalmente, os usuários assíduos e especialistas, são familiarizados com os aspectos sintáticos e semânticos do sistema e passam a demandar respostas mais rápidas e breves do sistema.

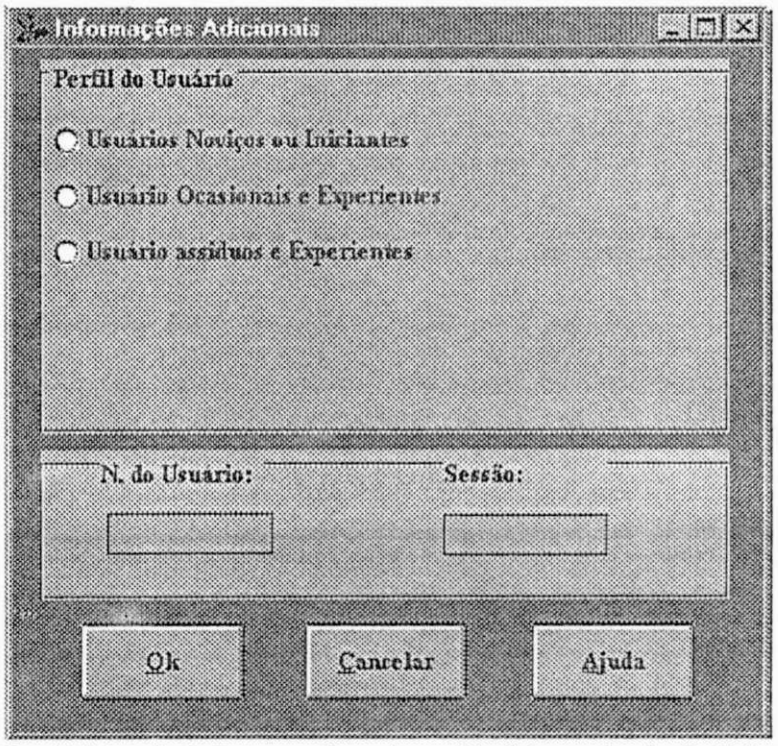



**Fig. 4.15 Janela de Informações sobre o Usuário e a Sessão de Coleta**

❑ Janela de Abertura de Arquivo - é uma caixa de diálogo que permite que o projetista selecione o nome do arquivo executável que contém a interface a ser avaliada (Seção 4.5.1.4);

❑ Janela de Informação sobre a Abertura do Arquivo - é a janela que contém informações sobre o resultado da operação de abertura do arquivo. Se a operação for bem sucedida será ativada a Janela de Entrada de Dados do Projeto, caso contrário a Janela de Abertura de Arquivo ficará ativa esperando uma intervenção do usuário (Seção 4.5.1.4).

❑ Janela de Confirmação da Coleta Dinâmica - permite que o projetista dê prosseguimento à coleta ou retorne à Janela de Informações Adicionais.

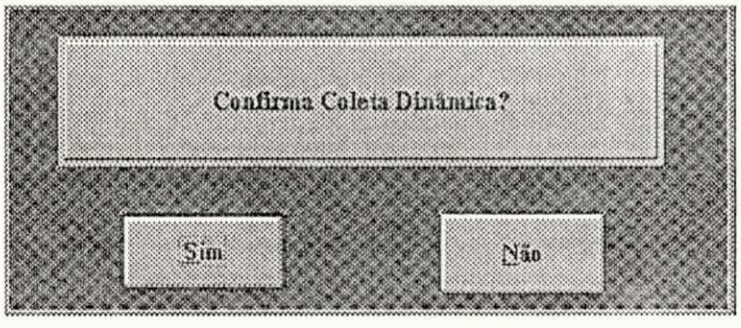



**Fig. 4.16 Janela de Confirmação da Coleta Dinâmica**

❑ Janela de Informações sobre a Ocorrência de Erros na Coleta - esta janela será mostrada caso haja algum erro durante a ativação da coleta dinâmica.


**Fig. 4.17 Janela de Informações sobre a Ocorrência de Erros na Coleta**

### *4.5.3 Resultados da Coleta Estática (RCE)*

Os resultados apresentados após a execução da coleta estática fornecem subsídios aos projetistas para análise da interface. Esta etapa consistirá da análise das

características visuais do protótipo frente a um conjunto de diretrizes de projeto de interface.

A apresentação dos dados coletados pode ser gráfica e/ou textual, colocando à disposição do projetista as informações necessárias para que ele avalie a qualidade da interface proposta. Estas informações consistem de: dados relativos à versão da interface sob análise e informações sobre as janelas que compõem a interface. As informações sobre as janelas são :

- ☑ Quantidade de janelas definidas;
- ☑ Os nomes das janelas;
- ☑ Quantidade de objetos em cada janela;
- ☑ Nomes dos objetos que compõem a janela;
- ☑ Os tipos de comandos existentes em cada janela;
- ☑ Os nomes dos objetos que contém as opções acima citadas;
- ☑ A quantidade de cores na janelas;
- ☑ Visualização das cores na janela;
- ☑ Visualização da cores dos objetos.

#### *4.5.3.1 Arquivos utilizados pelo módulo RCE*

Este módulo utiliza como entrada de dados os arquivos com extensão (.pje) e (.est), gerados durante a fase da coleta estática.

#### *4.5.3.2. Interface com o usuário do RCE*

A interface com o usuário do módulo de Resultados do Coletor Estático é composta das seguintes janelas:

- ✓ Janela de Abertura do Arquivo (Fig. 4.9);
- ✓ Janela de Informações sobre a Versão da Coleta (Fig. 4.18.);
- ✓ Janela de Resultados da Coleta Estática (Fig. 4.19)
- ✓ Janela de Informações sobre os Objetos em uma Janela (Fig. 4.20);
- ✓ Janela de Emissão das Cores em uma Janela (Fig. 4.21);
- ✓ Janela de Informações sobre os Comandos em uma Janela (Fig. 4.22).

A seguir são descritas as janelas que compõem a parte da interface referente ao módulo Resultado da Coleta Estática.

❑ Janela de Abertura de Arquivo - é uma caixa de diálogo que permite que o usuário selecione o nome do arquivo (.pje) criado na fase da Coleta Estática (Seção 4.5.1.4).

❑ Janela de Informação sobre a Ocorrência da Abertura - é a janela que contém informações sobre o resultado da operação de abertura do arquivo. Se a operação foi bem sucedida, após a liberação do arquivo, será ativada a Janela de Informações sobre a Versão da Coleta, caso contrário a Janela de Abertura de Arquivo ficará ativa esperando uma intervenção do usuário (Seção 4.5.1.4).

❑ Janela de Informação sobre a Versão da Coleta - é a janela onde o usuário informa a versão do projeto cujos resultados serão analisados. De posse dessa informação, o programa lerá o arquivo de projeto (.pje), até encontrar o registro que contém informações sobre a referida versão. Em seguida será lido o arquivo (.est), o qual contém as informações sobre a coleta e cujo nome encontra-se gravado no arquivo (.pje). Caso não exista um registro com a versão fornecida, será apresentada uma mensagem de erro.

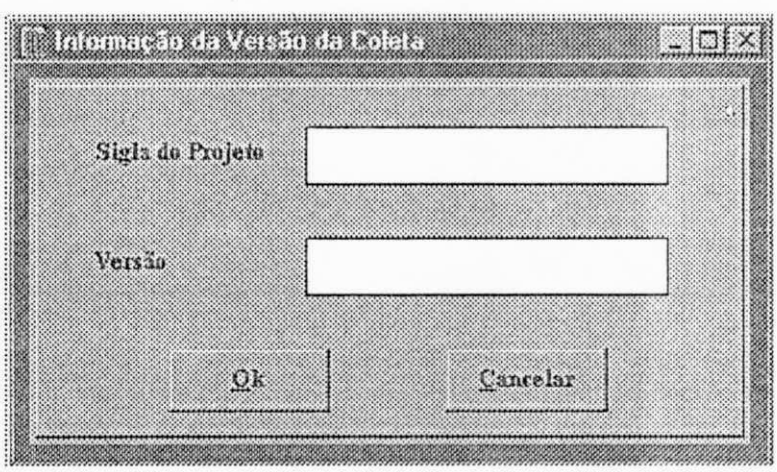



**Fig 4.18 Janela de Informação sobre aVersão da Coleta**

❑ Janela de Resultados da Coleta Estática - esta janela está dividida em duas partes. Na primeira parte são mostradas as seguintes informações:

☒ Nome do sistema;

☒ Nome do projeto;

☒ Nome do projetista;

☒ Sigla do sistema;

☒ Versão do protótipo a ser analisado;

☒ Data da coleta.

Na segunda parte encontram-se os botões de opção, para que o projetista selecione o resultado que lhe interessa:

☒ Informações sobre objetos na janela;

☒ Cores na janela;

☒ Comandos disponíveis;

☒ Encerrar emissão dos resultados.

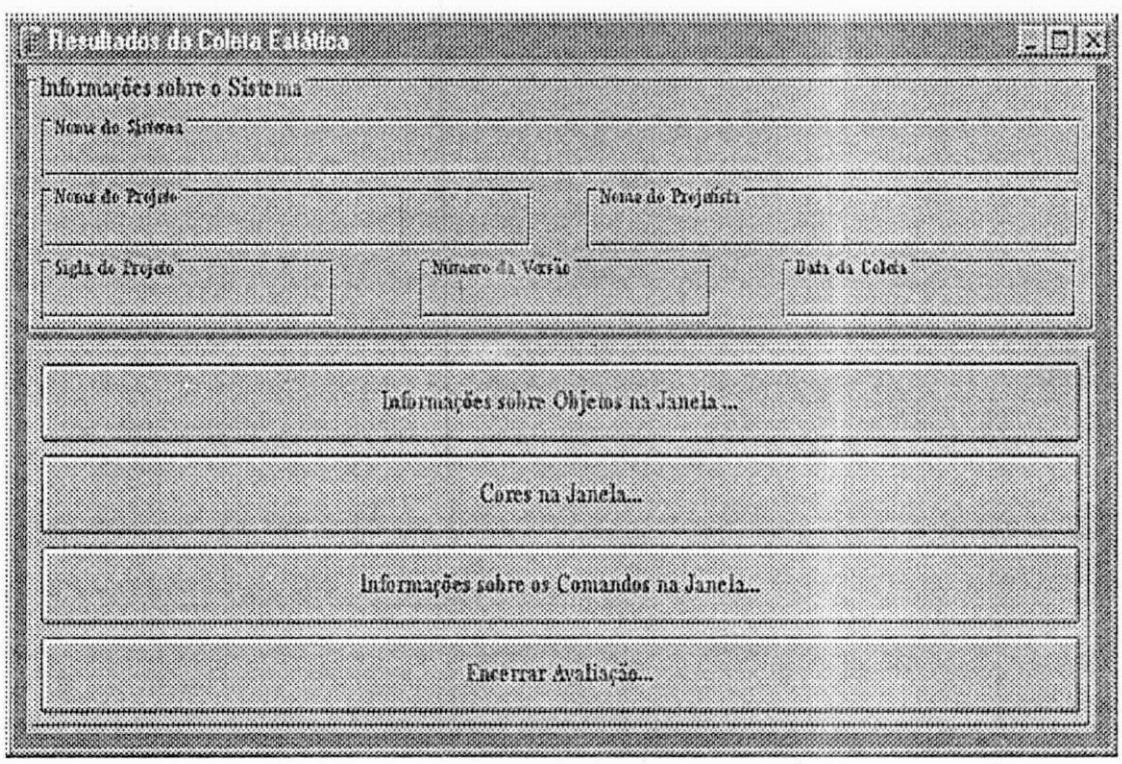



**Fig. 4.19 Janela de Resultados da Coleta Estática**

❑ Janela de Informações sobre os Objetos na Janela - esta janela contém informações sobre os objetos que compõem a interface:

☑ Quantidade de janelas;

☑ Os nomes das janelas;

☑ Quantidade de objetos em cada janela;

☑ Nomes dos objetos que compõem a janela;

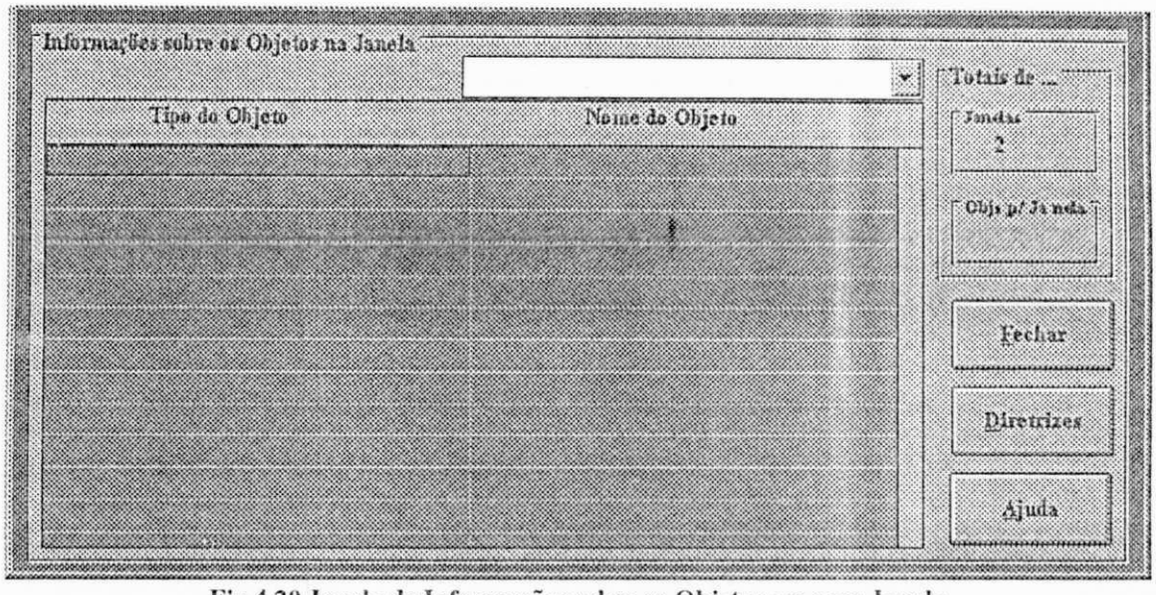



**Fig 4.20 Janela de Informações sobre os Objetos em uma Janela**

❑ Janela de Emissão das Cores - esta janela fornece ao projetista as seguintes informações:

☑ A quantidade de cores na janelas;
☑ Visualização das cores na janela;
☑ Visualização da cores dos objetos.

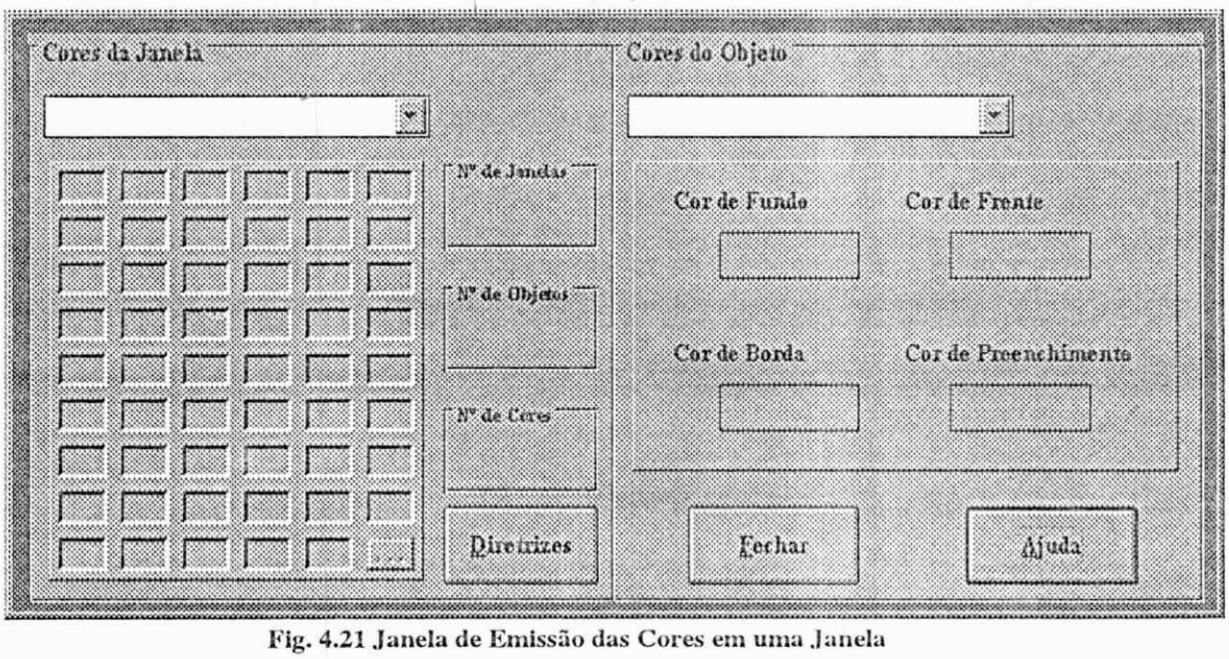



**Fig. 4.21 Janela de Emissão das Cores em uma Janela**

❑ Janela de Comandos Disponíveis - fornece ao projetista informações sobre:

☑ Os tipos de comandos existentes em cada janela;
☑ O nome dos objetos que contém os comandos mostrados.

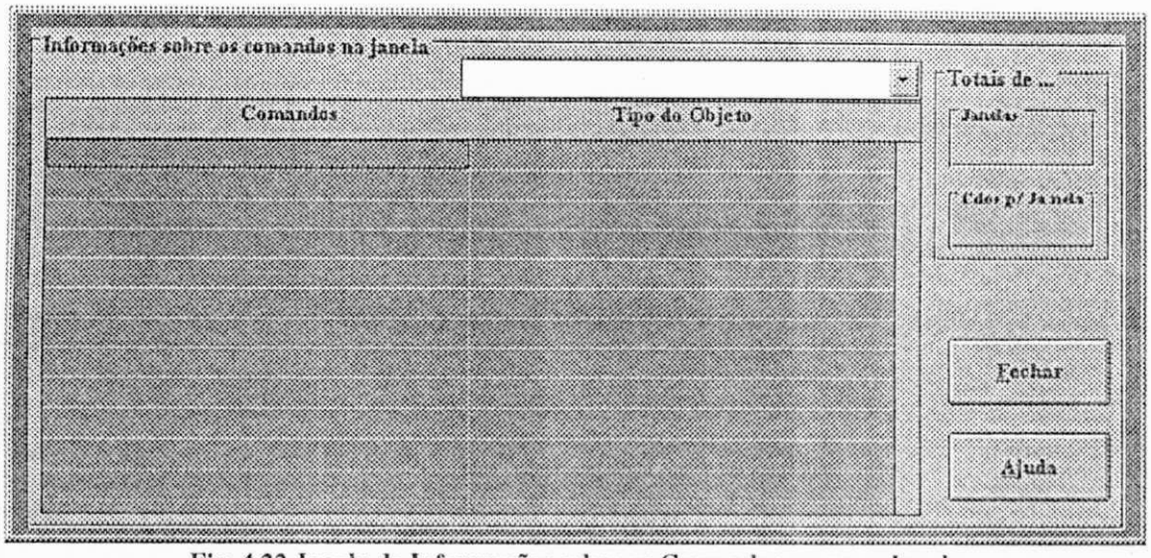



Fig. 4.22 Janela de Informações sobre os Comandos em uma Janela

### 4.5.4 Resultados da Coleta Dinâmica (RCD)

Este módulo apresenta os resultados da Coleta Dinâmica. Os resultados mostram o desempenho do usuário durante uma sessão de interação, levando em consideração o seu perfil.

#### 4.5.4.1 Arquivos utilizados pelo módulo RCD.

Os arquivos utilizados como entrada de dados no módulo RCD são os arquivos gerados durante a fase de interação do usuário com a interface HM (Coleta Dinâmica), são eles: (.pjd), (.din) e (.eve).

#### 4.5.4.2 Interface com o usuário do RCD.

A interface utilizada para apresentar os resultados da Coleta Dinâmica é composta das seguintes janelas:

- ✓ Janela de Abertura do Arquivo (Fig. 4.9);
- ✓ Janela de informações sobre o usuário, a versão e sessão da coleta (Fig. 4.23.);
- ✓ Janela de Resultados da Coleta Dinâmica (Fig. 4.24)
- ✓ Janela de Informações sobre uma Sessão (Fig. 4.25);
- ✓ Janela de Informações sobre uma Tarefa (Fig. 4.26);

✓ Janela de Informações sobre uma Janela (Fig. 4.27).

Descrição das janelas referentes ao módulo de Resultado da Coleta Dinâmica.

❑ Janela de Abertura de Arquivo - é uma caixa de diálogo que permite que o usuário selecione o nome do arquivo (.pjd) criado na fase da Coleta Dinâmica (Seção 4.5.1.4).

❑ Janela de Informações sobre a Ocorrência da Abertura - é a janela que contém informações sobre o resultado da operação de abertura do arquivo. Se a operação foi bem sucedida após sua liberação será ativada a Janela de Informações sobre o Usuário, a Versão e a Sessão da Coleta, caso contrário a Janela de Abertura de Arquivo ficará ativa esperando uma intervenção do usuário (Seção 4.5.1.4).

❑ Janela de Informações sobre o Usuário, a Versão e a Sessão da Coleta - é a janela que permite a seleção do arquivo para análise a partir do fornecimento da identificação do usuário, a versão do projeto e a sessão de coleta. De posse dessa informação o programa irá ler o arquivo de projeto (.pjd), até encontrar o registro que contém informação sobre o usuário, a versão e a sessão pretendidas. Em seguida serão lidos o arquivo (.din) e o arquivo (.eve), o quais contêm as informações sobre a coleta e cujos nomes encontram-se gravados no arquivo (.pjd). Caso não exista um registro com as informações fornecidas, será apresentada uma mensagem de erro.

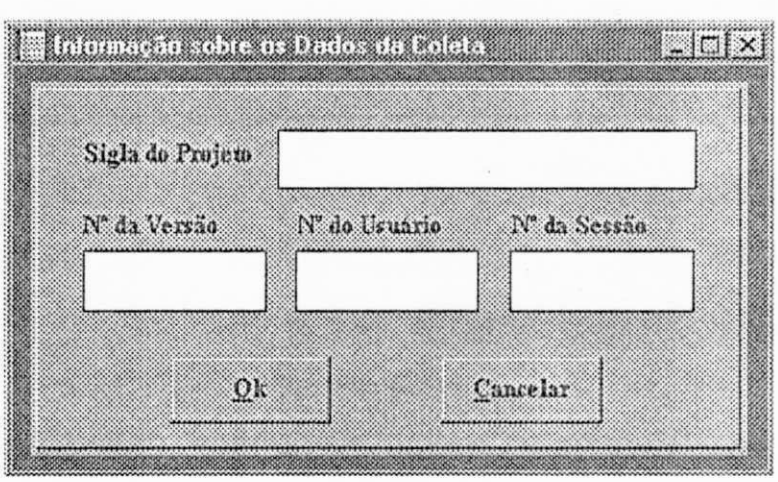



**Fig. 4.23 Janela de Informação sobre os Dados da Coleta**

❑ Janela de Resultados da Coleta Dinâmica - esta janela está dividida em duas partes. Na primeira parte são mostradas as informações sobre:

- ☒ Nome do sistema;
- ☒ Nome do projeto;
- ☒ Nome do projetista;
- ☒ Sigla do sistema;
- ☒ Versão do protótipo a ser analisado;
- ☒ Data da coleta.
- ☒ Número da Sessão.
- ☒ Número do usuário.
- ☒ Tipo do Usuário.

Na segunda parte encontram-se os botões de opção, para que o projetista selecione o resultado que lhe interessa:

- ☒ Informações sobre a Sessão.
- ☒ Informações sobre a Tarefa.
- ☒ Informações sobre a Janela.
- ☒ Encerrar apresentação dos resultados.

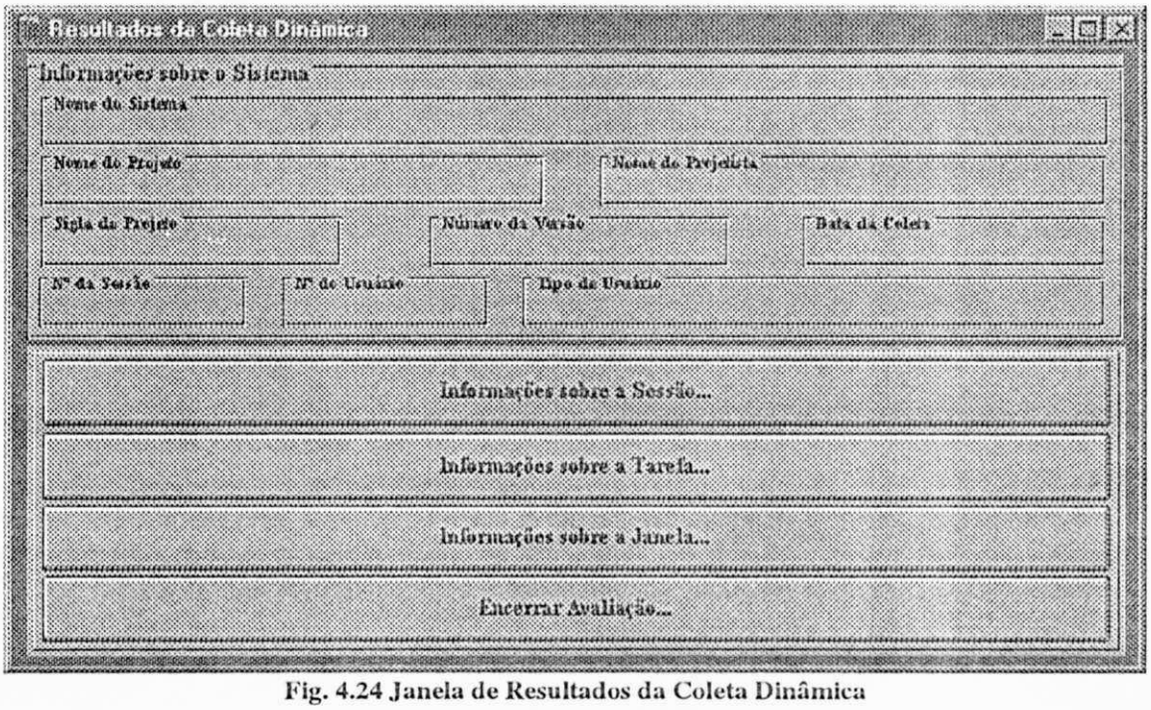



**Fig. 4.24 Janela de Resultados da Coleta Dinâmica**

❑ Janela de Informações sobre uma Sessão - esta janela contém informações sobre as ações executadas durante uma sessão:

- ☑ Duração da sessão.

- ☑ Quantidade de Erros na sessão.
- ☑ Comandos utilizados durante a Sessão, entre os disponíveis.
- ☑ Solicitação de ajuda (quantidade e duração).
- ☑ Quantidade de tarefas concluídas.
- ☑ Quantidade de tarefa abortadas.

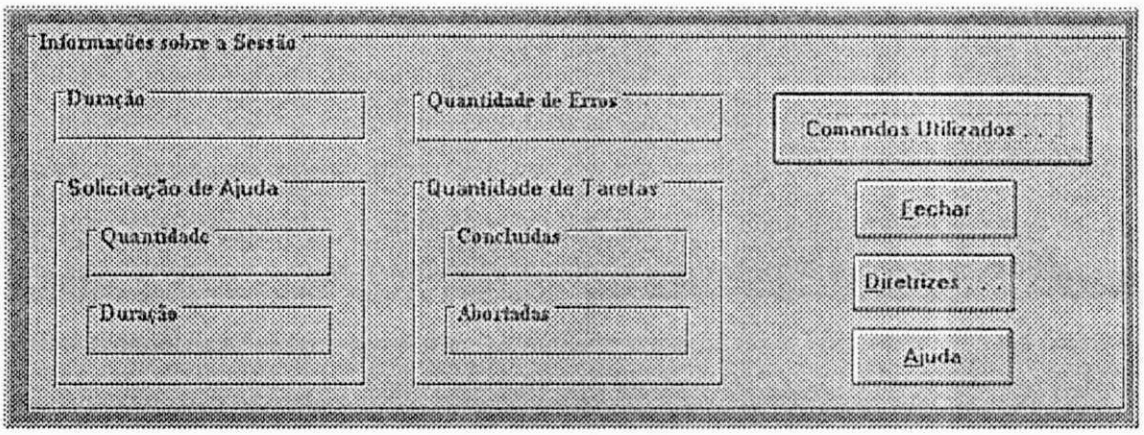



**Fig 4.25 Janela de Informações sobre a Sessão**

- ❑ Janela de Informações sobre uma Tarefa - esta janela fornece ao projetista as seguintes informações:
  - ☑ Percurso utilizado na execução de uma tarefa.
  - ☑ Duração
  - ☑ Quantidade de erros
  - ☑ Solicitação de ajuda (quantidade e duração).

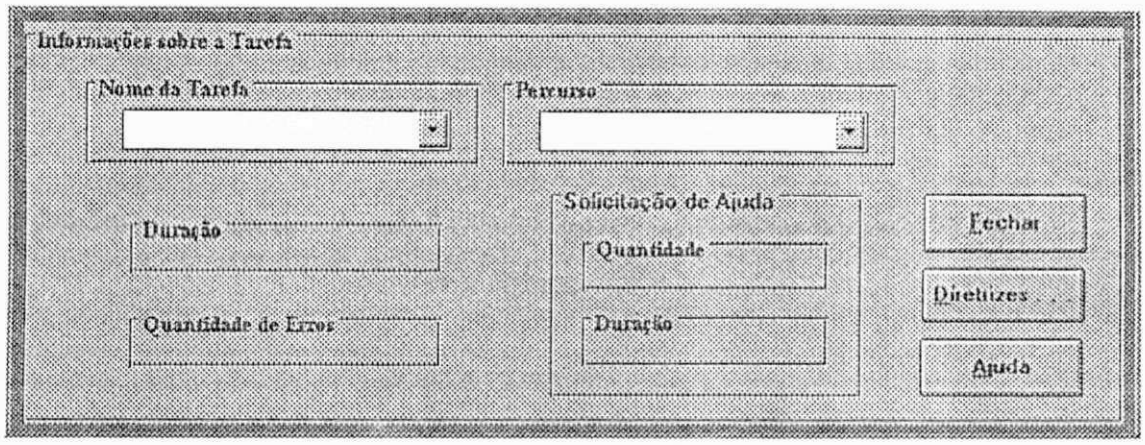



**Fig. 4.26 Janela de Informações sobre uma Tarefa**

- ❑ Janela de Informações sobre uma Janela - fornece ao projetista:
- ☑ Quantidade de erros
- ☑ Tempo de Permanência na janela.
- ☑ Menus apresentados na janela
- ☑ Solicitação de ajuda (quantidade e duração).

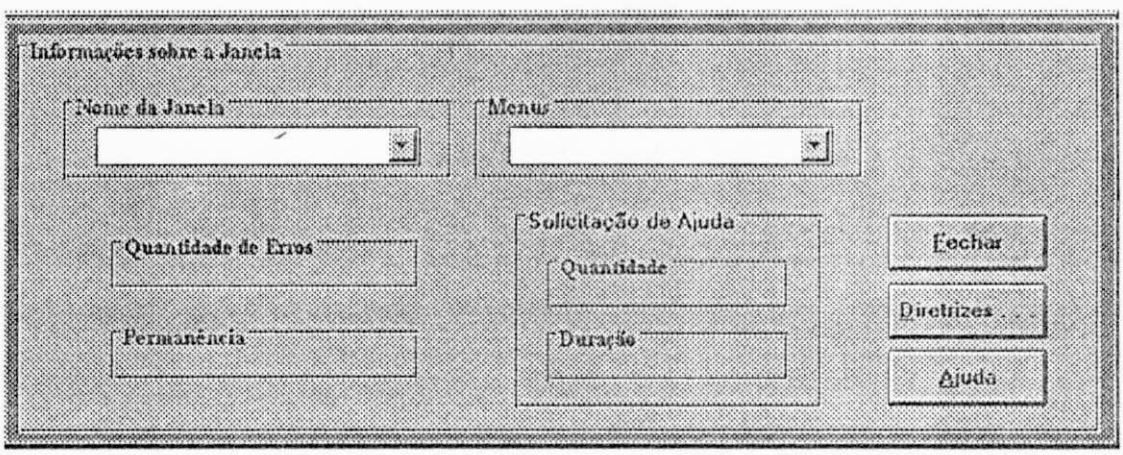



**Fig. 4.27 Janela de Informações sobre uma Janela**

### *4.5.5 Exclusão de Arquivos (EA).*

Este módulo foi construído para permitir que o projetista pudesse alterar os dados de formação do nome do arquivo da Coleta Estática e da Coleta Dinâmica. Também possibilita excluir um arquivo de coleta que já não lhe interesse mais.

### *4.5.5.1. Interface com o Usuário do módulo EA.*

A interface deste módulo é formada pelas seguintes janelas:

- ✓ Janela de Exclusão do Arquivo (Fig. 4.28 );
- ✓ Janela de Confirmação da Exclusão do Arquivo (Fig. 4.29).

❑ Janela de Exclusão do Arquivo - é uma caixa de diálogo que permite que o usuário selecione o nome do arquivo (.est ou din ou .eve), criado nas fases de Coleta Estática e Coleta Dinâmica, que ele deseja excluir. É permitido ao projetista excluir os arquivos de projetos (.pje e .pjd), só que para excluir estes arquivos é necessário que o projetista forneça a sua terminação. Isto é necessário porque esta terminação não se encontra disponível, uma vez que estes arquivos não estão diretamente ligados a um só arquivo (.est, ou .din, ou .eve). Na realidade eles contém informações sobre todas as versões e a sua exclusão terá que ser bem controlada para evitar que arquivos de coletas de outras versões fiquem indisponíveis. O botão **Ok** que se encontra na janela de diálogo fará com que a Janela de Confirmação da Exclusão seja ativada. Caso o projetista queira encerrar as exclusões, ele deverá pressionar o botão Cancelar, que o fará retornar a Janela Principal.

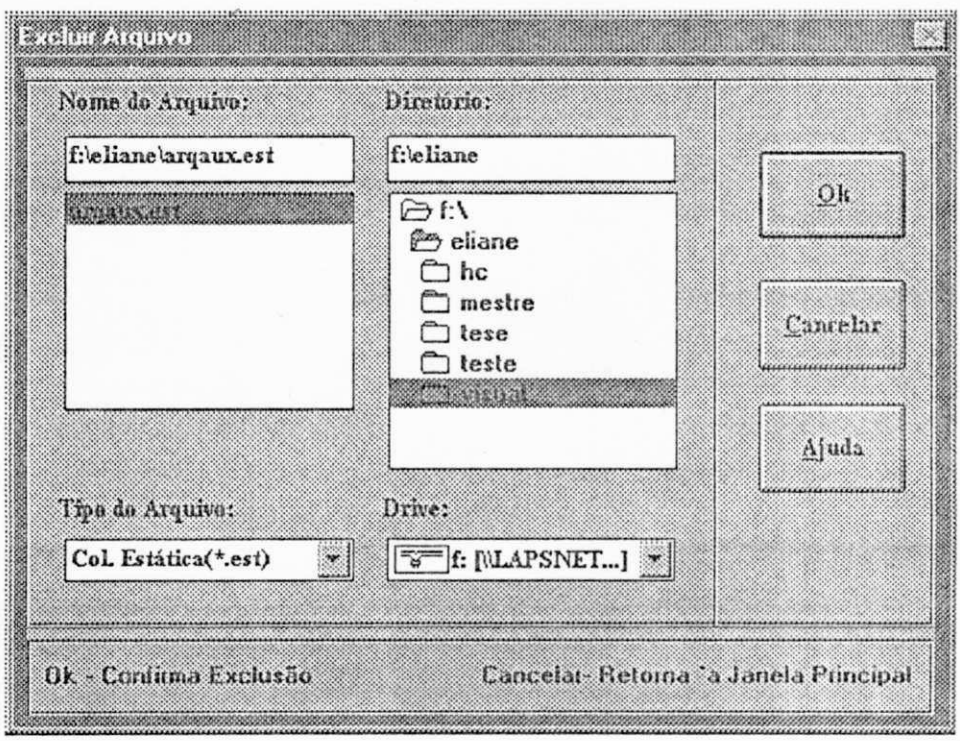



**Fig. 4.28 Janela de Exclusão de Arquivo**

❑ Janela de Confirmação da Exclusão do Arquivo - permite ao usuário após a seleção de um determinado arquivo, continuar a remoção ou desistir da exclusão, voltando a selecionar um novo arquivo.

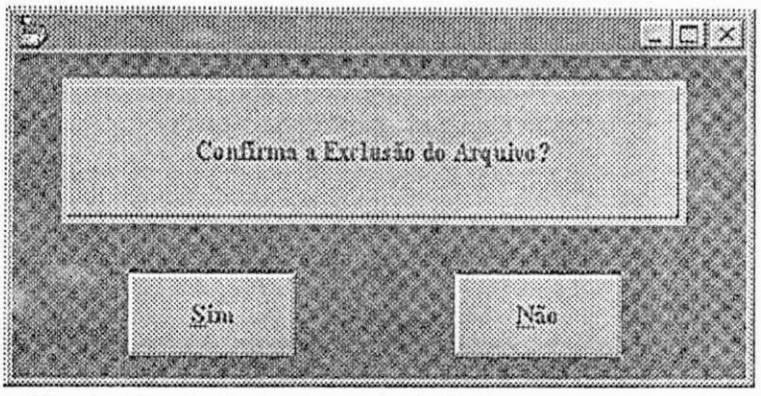



**Fig. 4.29 Janela de Confirmação da Exclusão de um Arquivo**

## *4.6. Ambiente de Desenvolvimento da* $FAI_{WIN}$

Para implementação da $FAI_{WIN}$ foi utilizada a ferramenta de programação *Visual Basic* da *Microsoft*, a qual, com seu mecanismo de programação orientada a eventos facilitou o processo de implementação.

Os demais módulos da $FAI_{WIN}$ foram implementados em C++, para ambiente *Windows*, utilizando as vantagens oferecidas por essa linguagem a nível de manipulação dos objetos e encapsulamento destes objetos em uma biblioteca de ligação dinâmica (*Dynamic link library* -DLL).

As DLLs são um dos elementos estruturais do *Windows*. Uma das principais vantagens da sua utilização é a independência entre aplicação e biblioteca, isto denota que a biblioteca pode ser alterada independentemente da aplicação que a utiliza. A ativação dos módulos internos da DLL ocorre através de troca de mensagens. O envio dessas mensagens depende do evento acionado pelo usuário quando da sua interação com a interface prototipada pelo *Visual Basic*. Maiores detalhes sobre o uso de DLLs encontram-se no apêndice A.

# Capítulo V

# Estudo de Caso - Bancada Virtual

Este capítulo apresenta os resultados de um estudo de caso realizado com o propósito de testar a funcionalidade da ferramenta $FAI_{WIN}$. Neste estudo foi utilizado o protótipo de uma interface de um sistema em desenvolvimento- a Bancada Virtual que consiste de um conjunto de equipamentos eletrônicos emulados no computador.

## 5.1 Descrição da Bancada Virtual.

Segundo Santos em [SANT96] o projeto da Bancada Virtual constitui uma alternativa para solucionar os transtornos causados pela escassez de equipamentos nos laboratórios dos Cursos de graduação em Engenharia Elétrica. Os instrumentos virtuais emulam os instrumentos reais, tanto na sua funcionalidade quanto na sua representação de controles e *displays* mas, com maior versatilidade.

A Bancada Virtual consiste em um conjunto de instrumentos de medição e geração de funções. A proposta de criação desta Bancada tem como propósito a aplicação didática e sua utilização tem como alvo principal os alunos do curso de graduação em Engenharia Elétrica, durante o treinamento laboratorial.

A programação visual dos instrumentos virtuais foi feita a partir da construção de uma biblioteca de objetos correspondentes aos elementos do painel de cada instrumento real, tais como mostradores, telas, botões e controles. A biblioteca foi construída utilizando-se o Visual Basic [SANT96].

Na Bancada Virtual a escolha do(s) instrumento(s) é feita na janela de menus, Fig. 5.1, onde se encontram a barra de menu e as barras de ícones.


**Fig. 5.1 Janela de menus para seleção de um instrumento virtual.**

A Fig. 5.2, apresenta um esquema com a seqüência de ações, que estão disponíveis na atual fase do projeto da bancada. Estas ações podem ser desencadeadas a partir da janela de menu, mostrada na Fig. 5.1

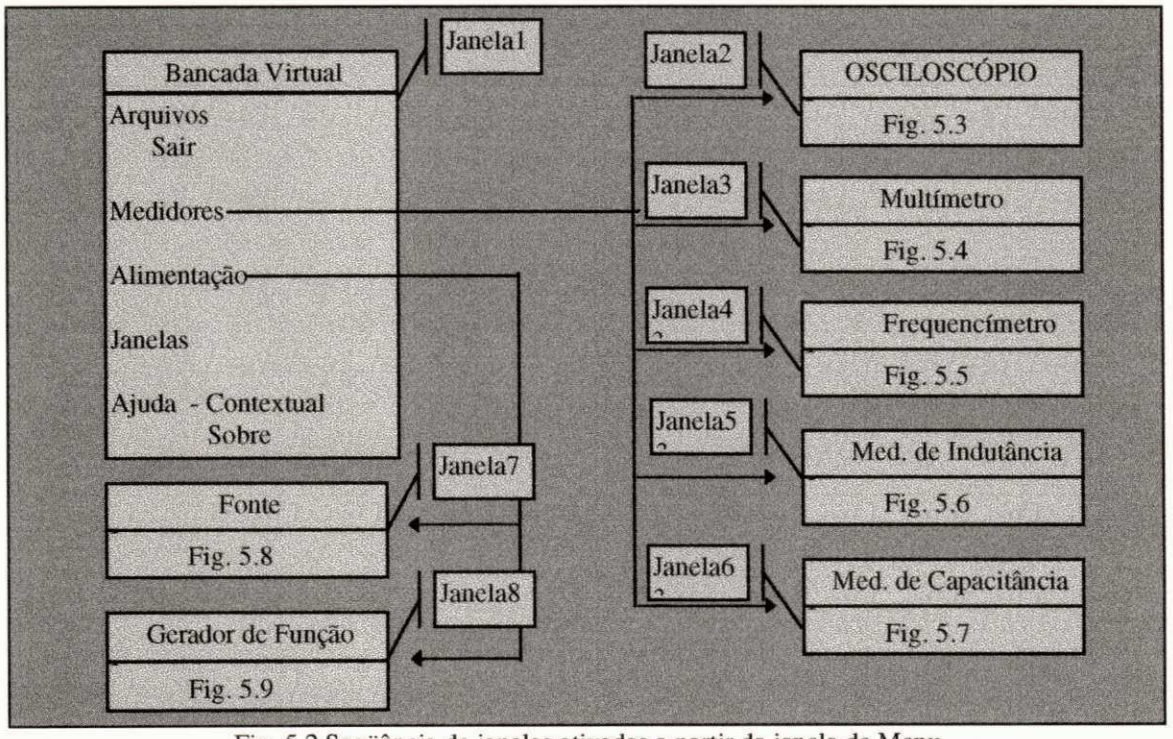



Fig. 5.2 Seqüência de janelas ativadas a partir da janela de Menu

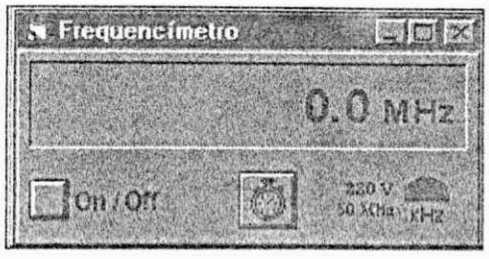



Fig 5.5 Frequencímetro Virtual

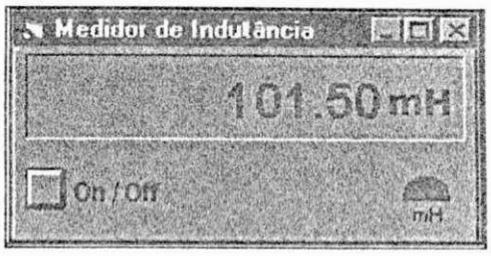



Fig 5.6 Medidor Virtual de Indutância

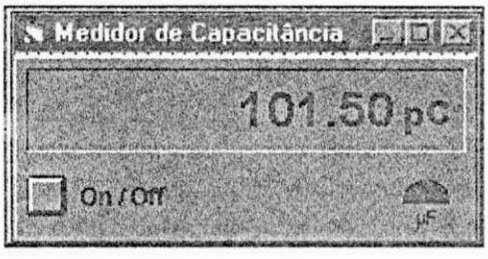



Fig 5.7 Medidor Virtual de Capacitância

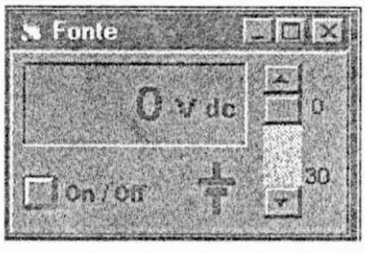



Fig 5.8 Fonte de Alimentação

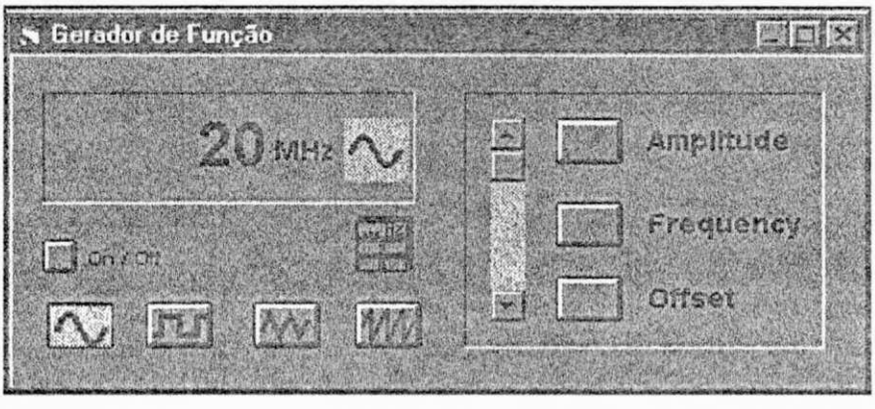



Fig. 5.9 Gerador de Funções.

## 5.2 *Planejamento do Experimento.*

O experimento teve como objetivo avaliar a adequação da interface aos seus usuários, comparando seu desempenho entre sessões e entre grupos, durante a realização de um mesmo conjunto de tarefas. Os usuários foram selecionados entre alunos de graduação e pós-graduação distribuidos em dois dos três grupos de usuários descritos na sessão 4.5.2.4. A descrição destes usuários é feita na sessão 5.2.2.1. Para tanto o experimento foi dividido em três etapas:

- Avaliação Estática;
- Coleta Dinâmica com os usuários e;

- ❑ Avaliação dos dados face às diretrizes.

### *5.2.1 Avaliação Estática.*

Inicialmente foi realizada a coleta estática (Capítulo IV.), quando foram coletadas as informações sobre os elementos que compõem o *layout* da interface tais como: cores, quantidades de objetos na janela, etc.

Os resultados dessa etapa são fornecidos através de três janelas, de acordo com a opção feita pelo projetista. As figuras abaixo mostram os resultados obtidos quando da seleção da janela principal ( Fig. 5.1).

A Fig. 5.10, mostra os resultados da coleta, relativos às informações sobre os objetos que compõem a interface: quantidade de janelas da interface, os objetos em cada janela, etc. A Fig. 5.11, apresenta a quantidade de cores por janela e quais são estas cores.

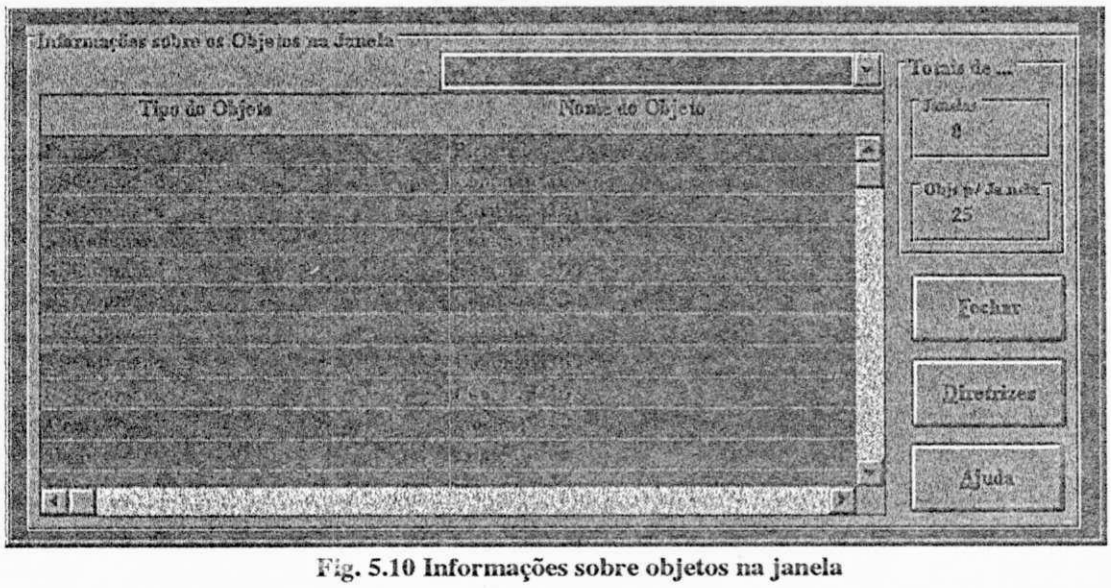



**Fig. 5.10 Informações sobre objetos na janela**

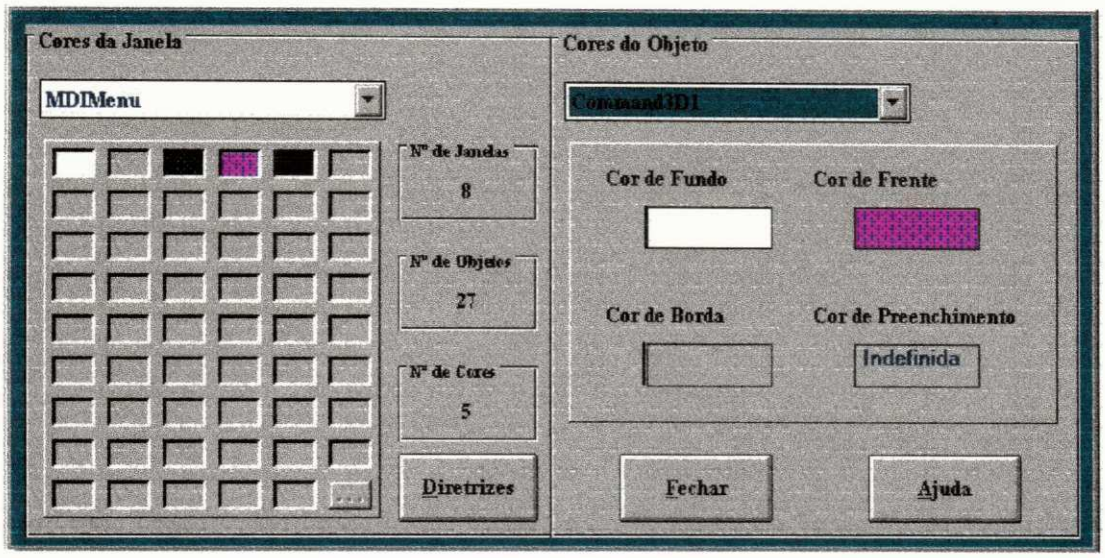



**Fig. 5.11 Informações sobre Cores na janela**

A Fig. 5.12 apresenta ao projetista, de modo discriminado, os comandos que compõem cada janela. Estes comandos poderão ativar as ações que darão continuidade à execução da tarefa que o usuário solicitou

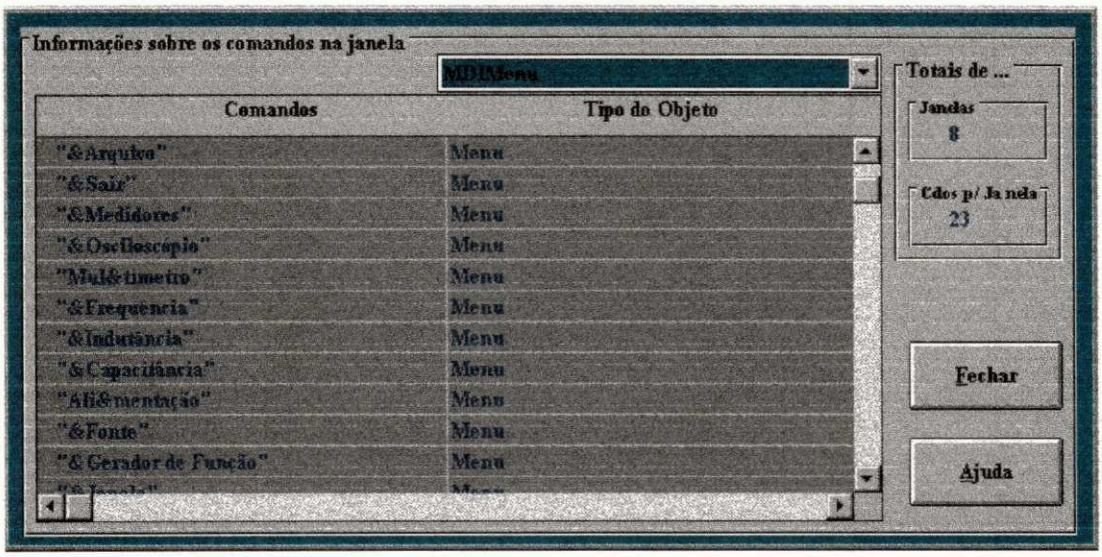



**Fig. 5.12 Informações sobre Comandos na janela**

### *5.2.2 Coleta Dinâmica.*

A segunda Etapa deste experimento consiste na Coleta Dinâmica dos dados a partir da interação com o usuário. A observação da interação tem como foco a realização de uma tarefa de acordo com a conceituação apresentada no capítulo anterior. Neste Experimento foram escolhidas duas tarefas:

1. A realização de uma medida através da utilização do osciloscópio e,
2. A seleção de um tipo de onda no Gerador de Funções.

A escolha dessas tarefas decorreu do fato das mesmas já se encontrarem em fase experimental no projeto de interface da Bancada Virtual.

#### *5.2.2.1 Coleta dos Dados.*

Esta etapa foi dividida em duas sessões para que o projetista pudesse fazer uma análise posterior comparando o desempenho dos usuários entre as sessões. Após delineadas as tarefas que fariam parte do experimento, foi selecionado um grupo de 14 usuários para realizar os testes.

Dentre os usuários, observou-se que seis (6) deles se enquadravam no perfil de usuários assíduos especialistas, os quais utilizam com freqüência os instrumentos reais que estão representados na Bancada Virtual e tem familiaridade com o uso de computadores. Os outros usuários (8) se enquadraram no grupo de usuários experientes (no uso de instrumentos) porém ocasionais.

Nas duas sessões de testes foi utilizada a mesma versão do protótipo, pois o objetivo dos testes não foi comparar versões de protótipos, mas comparar o desempenho dos usuários entre sessões. Dentre os aspectos investigados nestes testes destacam-se: a retenção de informações sobre os comandos disponíveis entre sessões, os comandos utilizados e o percurso efetuado na realização das tarefas.

### *5.2.3 Resultados da coleta*

Nesta sessão serão apresentados os resultados da coleta de dados efetuada nas duas sessões de avaliação. De posse destes dados o projetista terá subsídios para fazer uma avaliação heurística da interface, com base em um conjunto de diretrizes de projeto.

Inicialmente serão apresentados os resultados obtidos a partir da interação de usuários assíduos e especialistas.

❒ Na Fig. 5.13 é apresentado o percentual de tempo de permanência em cada janela durante uma sessão, levando em consideração a ordem em que foram ativadas.

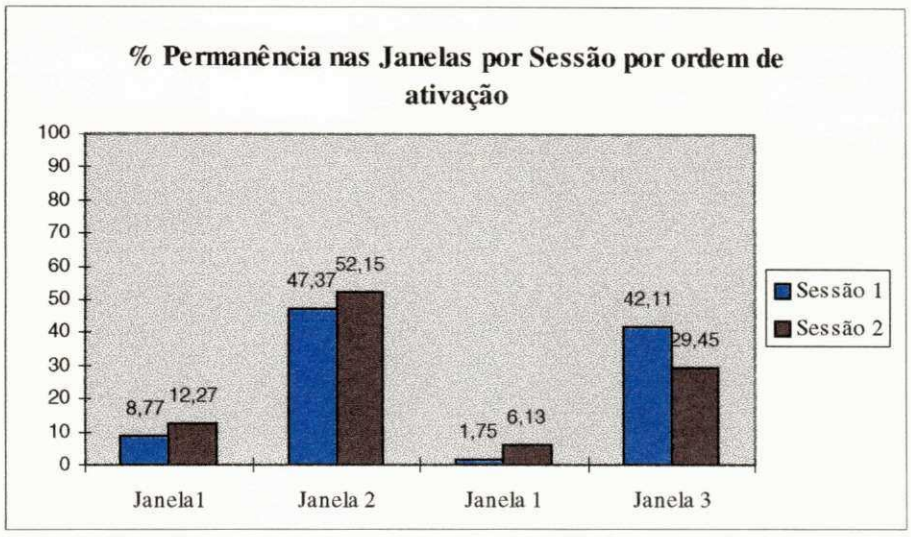



**Fig. 5.13 Percentual de tempo de permanência em cada janela durante a sessão**

❒ Percentual de tempo de permanência nas janelas ativadas para cada tarefa (Fig. 5.14). Este indicador pode auxiliar na restruturação de uma tarefa ou do conjunto de tarefas, em função do tempo de permanência nas janelas de interação.

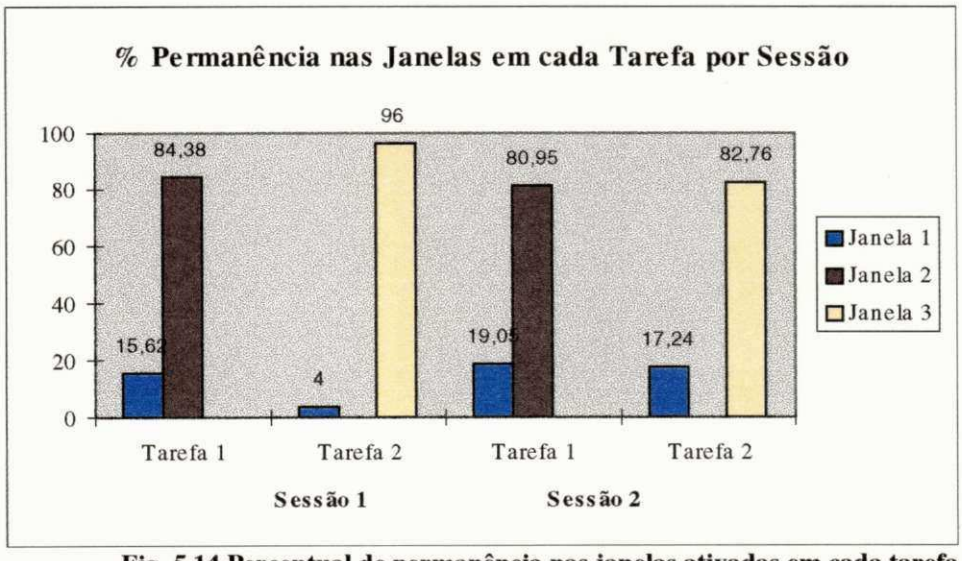



**Fig. 5.14 Percentual de permanência nas janelas ativadas em cada tarefa**

❒ O percentual do tempo gasto para realização das tarefas por sessão (Fig. 5.15). Este indicador pode auxiliar na escolha entre versões de um mesmo projeto, indicando a melhor escolha de dispositivo ou de estratégia de interação.

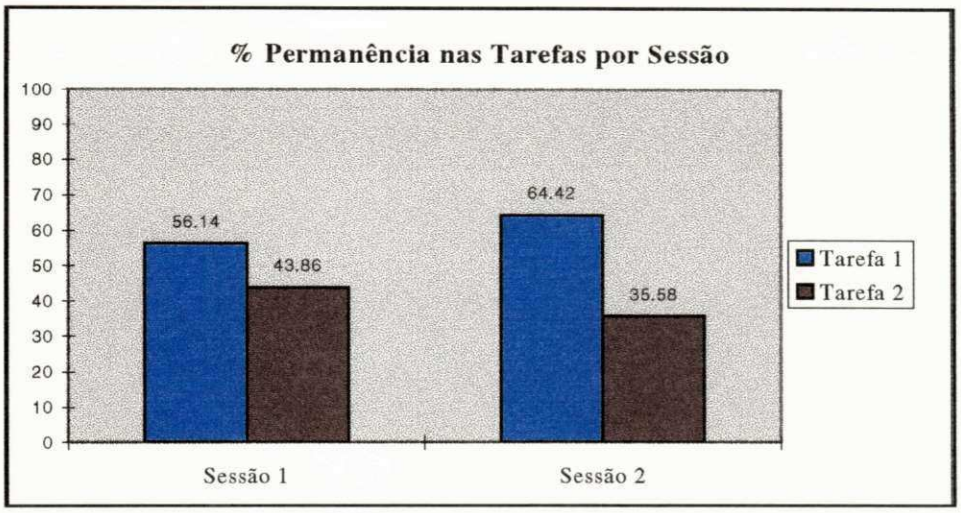



**Fig. 5.15 Percentual do tempo gasto para realização das tarefas por sessão**

As Figuras [Fig. 5.16, Fig. 5.17 e Fig. 5.18] mostram os resultados obtidos a partir da interação do grupo de usuários Ocasionais e experientes.

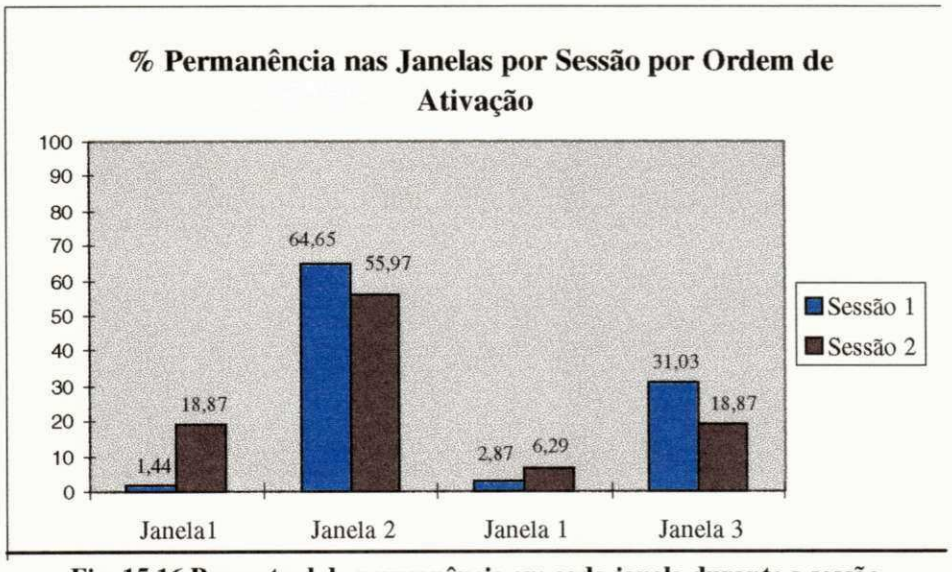



**Fig. 15.16 Percentual de permanência em cada janela durante a sessão**

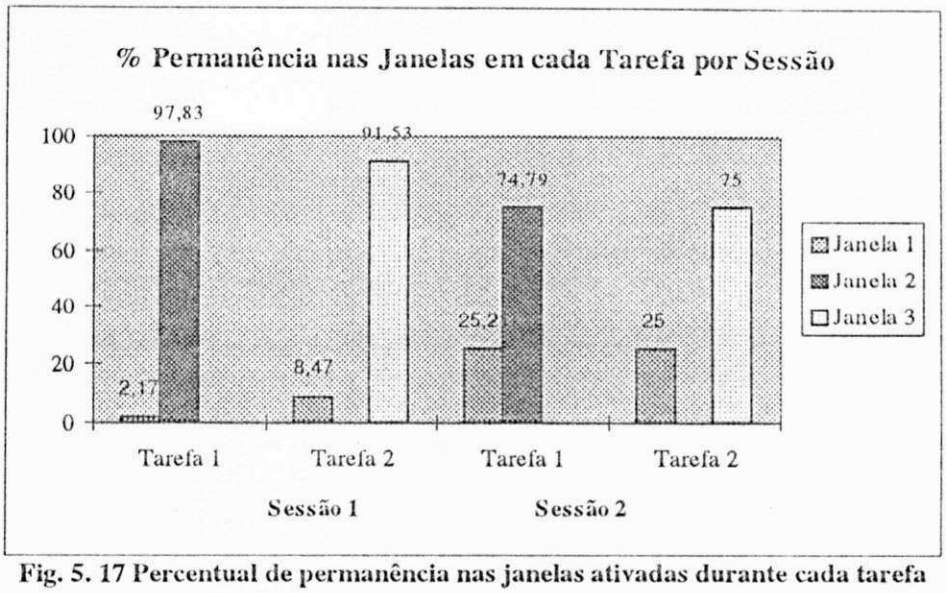



**Fig. 5. 17 Percentual de permanência nas janelas ativadas durante cada tarefa**

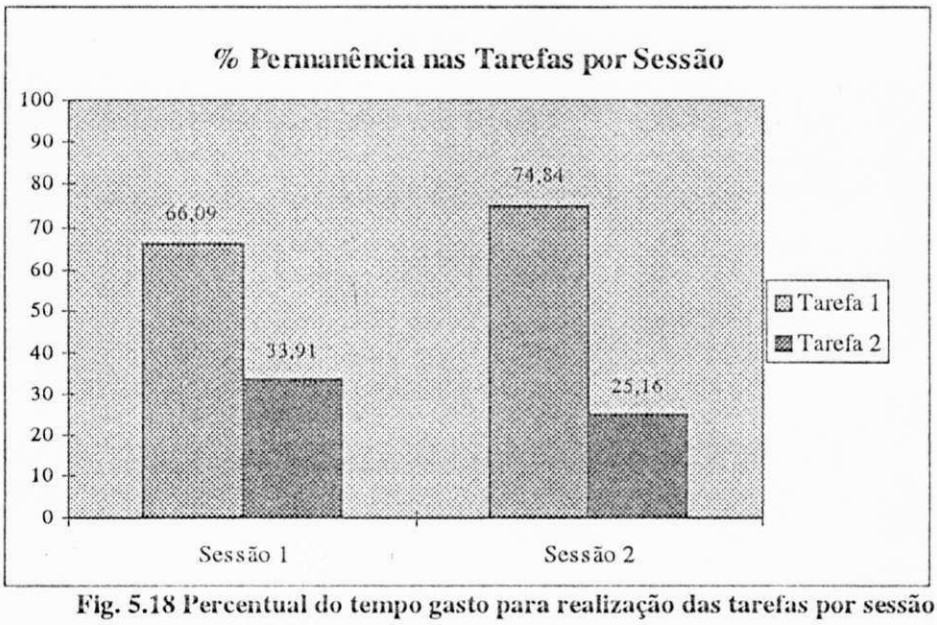



**Fig. 5.18 Percentual do tempo gasto para realização das tarefas por sessão**

### *5.2.4 Discussão dos Resultados das Coletas*

Os resultados aqui apresentados não podem ir além daqueles de um ensaio preliminar do funcionamento da ferramenta $FAI_{WIN}$, uma vez que o propósito do experimento foi ilustrar a funcionalidade da ferramenta de avaliação. No entanto, estes resultados se prestam para ilustrar o potencial da $FAI_{WIN}$, o que será feito nesta sessão na forma de comentários sobre o estudo de caso.

Os resultados da coleta estática, permitiram que o avaliador verificasse se o *layout* da interface, está de conformidade com as diretrizes genéricas de projeto da área. De

posse dessas informações ele pode avaliar e melhorar a aparência da interface, no que se refere, por exemplo ao uso de cores.

Os resultados da avaliação estática, ao serem fornecidos através das janelas da ferramenta facilitaram um confronto posterior com os comandos utilizados pelo usuário, mostrando assim quais os comandos que foram mais utilizados e aqueles que foram sub-utilizados. Desta forma a avaliação sobre a utilização dos comandos foi facilitada. De posse desta informação o projetista pôde avaliar a necessidade de reprojeto dos comandos ou de treinamento do usuário com o produto. Neste estudo de caso constatou-se a necessidade de uma ajuda on-line, ainda em desenvolvimento por ocasião da avaliação.

Os resultados apresentados sobre a quantidade e o tipo de "objetos apresentados em uma janela", facilitaram a verificação do uso da diretrizes de projeto relativas à quantidade recomendada de objetos nas janelas da bancada. Estes dados são úteis na identificação de janelas carregadas que causam fadiga e desorientação no usuário.

A coleta dinâmica foi executada em duas sessões. Os dados coletados forneceram ao avaliador informações sobre a sessão, a tarefa e sobre todas as janelas que foram ativadas durante as sessão com cada usuário. As informações obtidas sobre a interação tais como: a duração e a quantidade de ajuda solicitada pelo usuário, a quantidade de erros cometidos, a quantidade de tarefas concluídas e abortadas, e os comandos utilizados em cada sessão, permitiram uma análise criteriosa sobre a facilidade de uso do produto. Com estes dados é possível verificar o percurso que o usuário utilizou para executar determinada tarefa e observar se ele se adaptou melhor ao uso do mouse ou de teclas de atalho. As informações fornecidas através de gráficos facilitaram a comparação entre usuários e entre sessões.

Neste estudo de casos, uma das limitações dos resultados diz respeito ao reduzido número de usuários que se disponibilizaram a participar do experimento. Outra limitação foi a simplicidade das tarefas propostas aos usuários. O estágio de desenvolvimento do protótipo também contribuiu para a falta de aprofundamento nas tarefas, a exemplo da indisponibilidade do sistema de ajuda, impedindo por exemplo a monitoração de dados relativos às solicitações de ajuda. Portanto, a contribuição do processo de avaliação para o projeto da bancada foi insipiente, diante das limitações citadas.

Por outro lado, apesar das limitações citadas, este ensaio foi significativo enquanto teste preliminar da ferramenta $FAI_{WIN}$, uma vez que o seu objetivo de checar a

funcionalidade da ferramenta e ilustrar o seu funcionamento, foram atingidos apesar das limitações.

A versão atual da FAIwin, como já foi mencionado anteriormente, não objetiva fornecer um diagnóstico da avaliação de uma interface, mas registrar dados sobre a interação fornecendo subsídios para que os avaliadores possam emitir este diagnóstico. O estudo crítico desses dados permitirá que o avaliador possa no menor espaço de tempo possível, conhecer todas as vantagens e os possíveis gargalos do seu projeto de interface.

# Capítulo VI
# Conclusões

Este capítulo se propõe a apresentar as contribuições e limitações da ferramenta desenvolvida ao longo deste trabalho. A partir destas considerações serão propostas possíveis soluções para as limitações encontradas, na forma de sugestões para trabalhos futuros.

## 6.1 Discussão dos Objetivos Atingidos no Trabalho

Dentre os objetivos propostos no início deste trabalho, acredita-se que a proposta de auxiliar na avaliação de diferentes aspectos das interfaces prototipadas em ambiente *Windows*", foi alcançada. A ferramenta na sua versão atual consegue, através de seus dois módulos de coleta, capturar a maioria das informações necessárias ao avaliador de interface, tanto do ponto de vista de sua proposta original, quanto do ponto de vista da literatura.

Um outro objetivo estabelecido foi o de "assegurar a generalidade da ferramenta de modo a poder avaliar interfaces geradas para ambiente *Windows* por diferentes ferramentas". Este objetivo foi parcialmente alcançado uma vez que esta independência se limitou ao coletor dinâmico, como será discutido a seguir. O coletor estático, por sua vez, está limitado ao prototipador Visual Basic e às estruturas por ele geradas.

Do ponto de vista da coleta dinâmica, a coleta de informações independe da ferramenta que gerou a interface, uma vez que os dados são capturados através do mecanismo interno do *Windows -hooks*. Isto é possível uma vez que a ferramenta

FAI$_{WIN}$ "roda" em *background*. Esta estratégia também possibilitou uma coleta de forma transparente para o usuário.

Portanto, as informações obtidas no processo avaliatório são mais confiáveis, uma vez que o usuário livre de pressões impostas pela observação explicita, interagirá de modo mais natural, deixando vir à tona todas as dificuldades da interação que de outro modo poderiam ter sido inibidas. Em uma avaliação com observação explícita, os usuários ficam relutantes em ajudar ao projetista a descobrir os gargalos do projeto da interface, com receio que o seu próprio desempenho seja o objeto da avaliação.

Finalmente, quanto ao objetivo de "fornecer dados de forma clara e objetiva, para o projetista", também acredita-se que tenha sido alcançado. Se comparada a interfaces de outras ferramentas de coleta de dados descritas na literatura, que consistem basicamente de entrada a partir de linguagem de comandos e saídas na forma de tabelas, a FAI$_{WIN}$ atingiu plenamente este objetivo ao utilizar os recursos do ambiente *Windows*.

## *6.2 Limitações da Ferramenta*

Como já foi mencionado, na sua versão atual, o coletor de dados estático da FAI$_{WIN}$ está limitado a interfaces geradas pelo *software Visual Basic*. Esta limitação resultou da variedade de formatos de arquivo gerados por diferentes ferramentas. Para que fosse possível coletar as informações sobre as propriedades de todos os objetos de uma interface nestes diferentes arquivos, seria necessário um estudo aprofundado de seus respectivos formatos, tarefa que demandaria tempo e esforço incompatíveis com o escopo deste trabalho. É necessário salientar que estes arquivos estão gravados em ASCII, e que os manuais das ferramentas que os criam não detalham o seu conteúdo.

Uma outra limitação da atual versão da FAI$_{WIN}$, diz respeito à apresentação dos resultados da coleta na forma de gráficos, com recursos internos à ferramenta. Para compensar esta limitação foi gerado um arquivo texto, e a partir da utilização de uma planilha eletrônica, como a *Excel*, estes gráficos podem ser construídos.

## 6.3 Contribuições Esperadas

Acredita-se que a principal contribuição deste trabalho tenha sido a construção da ferramenta $FAI_{WIN}$. Como conseqüência, acredita-se que a partir das facilidades oferecidas por ela, os projetistas sentir-se-ão motivados a analisar suas propostas de interface.

## 6.4 Propostas de Continuidade

Dando continuidade ao desenvolvimento da ferramenta $FAI_{WIN}$, será necessário o desenvolvimento de alguns aspectos e a inclusão de outros, dentre os quais pode-se destacar:

- Inclusão do módulo de geração de gráficos (de barra, de pizza, etc.) no ambiente da ferramenta, para apresentação dos resultados da coleta.
- Expansão do Coletor de Dados Dinâmicos para coletar tempos de resposta do sistema em relação a uma ação do usuário.
- Tornar o Coletor de Dados Estáticos independente da ferramenta que gerou a interface sob avaliação de modo que possa avaliar protótipos gerados em outras ferramentas além do *Visual Basic*, tais como: *Visual C++, Delphi*, etc.
- Expandir o conjunto de dados coletados pela $FAI_{WIN}$, de modo que ela possa auxiliar na avaliação de um conjunto mais abrangente de fatores de qualidade.
- Expandir o módulo de apresentação dos resultados das coletas acrescentando recursos para análise estatística dos dados, de modo a tornar estas informações ainda mais consistentes.

Certamente, que ainda há muito espaço para expansão da ferramenta $FAI_{WIN}$, particularmente com a constante inclusão de recursos de construção de interfaces no ambiente alvo, porém espera-se que esta versão já contribua com a elevação da qualidade dos produtos gerados no ambiente para o qual foi proposta.

# *REFERÊNCIAS BIBLIOGRÁFICAS*

[ABRA77] ABRAMS, Marshall D.,TREU, Siegfried. A Methodology for Interactive Computer Service Measurement. Communications of the ACM, New York-USA, v.20, n.1, p.936-944, Dec. 1977.

[ALSI93] ALSINA, Marcos Sebastian. Programação em Windows - Infocon. Campina Grande-PB: UFPB/COPIN, 1993. 73p. (Notas de Aula).

[ALSI94] ALSINA, Marcos Sebastian. Netcomm uma Ferramenta para o Desenvolvimento de Aplicações Multimídia Distribuídas e Gerência de Redes em Ambiente Windows. Campina Grande-PB: Universidade Federal da Paraíba, 1994. 109p. : (Dissertação, Mestrado em Eng. Elétrica)

[BOUC92] BOUCHERAT, Peter. Evaluation case study I; Adaptive Intelligent Dialogues. In: DOWNTON, Andy. Engineering the Human-Computer Interface. Singapura: McGraw-Hill Book Company, 1992. Cap. 13, p.356-383.

[BOYC92] BOYCE, Jim, et al. Por Dentro do Windows 3.1; Guia completo. Tradução por Eloísa Beatriz Ciarelli. Rio de Janeiro: Ciência Moderna, 1992. 527p.

[CALV94] CALVERT, Charles. Programando Aplicações em Windows com C & C++. Tradução por Otávio Tannus. Rio de Janeiro: Berkeley, 1994. 710p.

[CHRI94] CHRISTIAN, Kaare. COMO FUNCIONA O WINDOWS™. São Paulo: Editora Quark do Brasil Ltda, 1994. 201p.

[COHI83] COHILL, A. M.,EHRICH, R. W. Automated tools for the study of human/computer interaction. In Proceedings of Human Factors Society 27 th annual Meeting. Human Factors Society, Norfolk-Va, p.897-900, Oct. 1983.

[COX87] COX, Brad J. Object Oriented Programming. USA: Addison-Wesley Publishing Company, 1987. 274p.

[COX93] COX, Kevin, WALKER, David. User-Interface Design. 2.ed. Singapura: Editora Prentice Hall, 1993. Cap.3. p.80-119.

[DESO89] DESOY, Jonh F., LIVELY, William M., SHEPPARD, Sally V. Graphical Specification of User Interfaces with Behaviour Abstraction. Human Factors in Computing System-CHI'89, Texas-USA, p.139-144, May 1989.

[DOWN88] DOWNTON, Andy. Engineering the Human-Computer Interface. Singapura: McGraw-Hill Book Company, 1992. Cap. 12. p. 325-355.

[ESTE90] ESTEVAM, R. DE C. Estudo sobre Desenvolvimento de Interface: Definição de Técnicas e de Avaliação baseadas na Satisfação do Usuário. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 1990. (Dissertação, Mestrado em Informática).

[GURE95] GUREWICH, Nathan. Visual Basic 3.0 for Windows-Guia do Programador. Tradução por André Rebelo Costa. Rio de Janeiro: Berkeley, 1995. 747p.

[HALL93] HALLIDAY, Carolina M. Segredo do Pc. Tradução Claudio Lobo. Rio de Janeiro: Berkeley Brasil Ed, 1993. 300p.

JACOB, Robert J.K.. The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look At Is What You Get. ACM Transactions on Information Systems, USA, n.9, p.152-169, Apr. 1991.

[HANS84] HANSON, S. J, KRAUT, R. E., FARBER, J. M. Interface design and multivariate analyses of UNIX command use. ACM Transactions off Information Systems, USA, v. 2, n.1, p.42-57, 1984.

[HAYE89] HAYES, E. Baran. A Guide to GUIS. Byte., USA, July. 1989.

[LEA88] LEA, Martin. Evaluating user interface designs. In: RUBIN, Tony. User Interface Design for Compute Systems. Chichester: John Wiley & Sons (Ellis Horwood Limited), 1988. Cap. 7, p. 134-167.

[MICR92A] MICROSOFT CORPORATION. Microsoft® Windows 3.1 Programming Tools. Redmond-USA: Microsoft Press, 1992. 265p.

[MICR92B] MICROSOFT CORPORATION. Microsoft® Windows 3.1 Guide to Programming. Redmond-USA: Microsoft Press, 1992. 567p.

[MICR92C] MICROSOFT CORPORATION. Microsoft® Windows 3.1 Programmer's Reference. Redmond-USA: Microsoft Press, 1992. 4.v.

[MICR93A] MICROSOFT CORPORATION. MICROSOFT® VISUAL BASIC™ Programmer's Guide; Programming System for Windows™: Version 3.0. Redmond-USA: Microsoft Press, 1993. 711p.

[MICR93B] MICROSOFT CORPORATION. Microsoft® Visual Basic™ Language Reference; Programming System for Windows™ Version 3.0. Redmond-USA: Microsoft Press, 1993. 674p.

[MICR95] MICROSOFT CORPORATION. Microsoft® Visual C++™ Getting Started; Professional System for Windows™ Version 1.5. Redmond-USA: Microsoft Press, 1995. 49p.

[MIKE93] MIKES, Steven. Unix para Usuários e Programadores do MS-DOS. Tradução por. Fernando Cabral. Rio de janeiro: Campus, 1993.

[MIZ190] MIZRAHI, Victorine Viviane. Treinamento em linguagem C: Curso Completo - Modulo I. São Paulo: McGraw-Hill, 1990. 241p.

[MIZ290] MIZRAHI, Victorine Viviane. Treinamento em linguagem C: Curso Completo - Modulo I.I. São Paulo: McGraw-Hill, 1990. 273p.

[MOTA93] MOTA, Edjair de Souza. $H_2o$ - Um Ambiente Gráfico para Avaliação de Desempenho. Campina Grande-PB: Universidade Federal da Paraiba, 1993.136p. (Dissertação, Mestrado em Informática).

[MOUR92] MOURA, Antão. Fundamentação para Engenharia de Software. Campina Grande-PB: UFPB/COPIN, 1992. 67Pp.(Notas de Aulas).

[MUNI79] MUNIPOV, V. M.. Ergonomics as a Factor in Social and Economic Development, Ergonomics, USA, n.22, p.607-611, June. 1979.

[NAKA95] NAKAYAMA, Lauro. Ferramenta de Avaliação de Interfaces com Usuário no Ambiente AGILE. Campina Grande-PB: Universidade Federal da Paraiba, 1995. (Dissertação, Mestrado em Eng. Elétrica)

[NEAL84] NEAL, A. S., SIMONS, R.M. Playback: A method for evaluating the usability of software and is document. IBM Systems Journal, USA, V.23, N.1, p.82-95. 1984.

[NEIB93] NEIBAUER, Alan, R. O ABC do Windows 3.1. Tradução por Elaine Somma Andrade Pezzoli; revisão técnica por Telma Marcela Salviati. São Paulo: McGraw-Hill Ltda, 1993. 284p.

[NELS94] NELSON, Ross. Visual Basic For Windows.. Tradução por Lars Gustav Erick Unonius; Rev. Técnica por Antônio Brás Uchoa. São Paulo: Makron Books, 1994. v.3.

[NOCN89] NORCIO, ANTHONY F., STANLEY, Jaki. Adaptive Human-Compute Interfaces: A literature Survey And Perspective, IEEE Transactions on Software Engineering, USA, n.19, p.399-408, Apr. 1989.

[NORT93] NORTON, Peter. Desvendando O Pc E Ps/2 /Peter Norton. Tradução por Daniel Vieira. 4 ed. Rio de Janeiro: Editora Campus, 1993. 394p.

[PAPP91] PAPPAS, Chis H., MURRAY, William, H. Turbo C++ Completo e Total. Tradução por Mario Moro Fecchio; revisão técnica por José Eduardo Maluf de Carvalho. São Paulo: Makron, McGraw_Hill, 1991. 771p.

[PERE94] PEREIRA, Suely Fernandes Torres. Especificação E Implementação De Ambiente Gráfico Para Sitim-150. Campina Grande-PB: UFPB/COPELE, 1994. (Relatório, Estágio Supervisionado - Engenharia Elétrica).

[PETZ93] PETZOLD, CHARLES.Programando Para Windows 3.1;Edição autorizada da Microsoft. Tradução e revisão técnica por Jeremias R. D. Pereira dos Santos. São Paulo: MAKRON Books, 1993. 1033p.

[PROC92] PROCÓPIO, C.D. Desenvolvimento Do Gerenciador De Diálogos E De Ferramentas De Prototipagem Do Sistema Agile.Campina Grande-PB: Universidade Federal da Paraiba, 1992. (Dissertação, Mestrado em Informática)

[QUEI94] QUEIROZ, José E. Rangel. Validação de uma Metodologia de Projeto de Interfaces Usuário-Computador. Campina Grande-PB: Universidade Federal da Paraiba, 1994. (Dissertação, Mestrado em Eng. Elétrica)

[RATH93] RATHBONE, Andy. Windows 3.1 Para Leigos. Tradução por Claudio Costa. Rio de Janeiro: Berkeley Brasil Ed, 1993. 300p.

[RICH92] RICHTER, Jeffrey M. Windows 3.1: A Developer's Guide. 2ed. San Mateo-USA: M&T Books, 1992. 713p.

[RUBE95] RUBENKING, Neil J.. Programação em Delphi para Leigos. Tradução por Ana Beatriz Tavares dos Santos Pereira. São Paulo: Berkeley, 1995. 372p.

[SANT92] SANTOS Jr, Rinaldo. Módulo: Gerenciador de Telas e Gerenciador de Periféricos de um Sistema de Prototipagem Rápida Usuário Computador. Campina Grande-PB: Universidade Federal da Paraiba, 1992. (Dissertação, Mestrado em Eng. Elétrica).

[SANT96] SANTOS , Serrano André. Desenvolvimento de Interfaces Gráficas para Instrumentos Virtuais. Campina Grande-PB Universidade Federal da Paraiba, 1996. (Relatório Técnico, Projeto PIBIC).

[SARA92] SARAN, Rogério. Programando em Windows 3.1. São Paulo: Érica, 1992. 393p.

[SCHI91] SCHILDT, Herbert. C++ Completo e Total. Tradução por Marcos Ricardo Alcântara Morais, revisão técnica por José Renato Adorni Martins e Roberto Carolos Mayer. São Paulo: Makron Books, McGraw-Hill, 1991. 889p.

[SCHI92] SCHILDT, Herbert. Turbo C++ Guia do Usuário/Herbert Schildt: Tradução e revisão técnica por Pedro Manfredi Jc. São Paulo: Makron Books, McGraw-Hill, 1992. 592p.

[SHNE83] SHNEIDERMAN, Ben. Direct Manipulation: a Step Beyond Programming Languages. IEEE Computer, USA, n.16, p.37-69, Aug. 1983.

[SHNE92] SHNEIDERMAN, Ben. Designing the User Interface: Strategies for Effective Human-Computer Interaction., 2nd Ed. USA: Addison-Wesley Publishing Company-Reading, 1992, 573p.

[SIL194] SILVA, Edlange Lopes da. Expansão do Sistema Agile e Migração para Ambiente Unix. Campina Grande-PB: UFPB/COPELE, 1994. 72p (Relatório técnico).

[SIL292] SILVA FILHO, José Bezerra. Avaliação Qualitativa de Interfaces Bancárias; com Proposta de Diretrizes de Projeto. Campina Grande-PB: Universidade Federal da Paraiba, 1992. (Dissertação, Mestrado em Informática)

[SIL393] SILVA, Lilian Gibson. UNIFIC: Uma API para a Unificação da Programação entre Diferentes GUIS. Campina Grande-PB: Universidade Federal da Paraiba, 1993. 192p. (Dissertação, Mestrado em Informática).

[SIO191] SIOCHI, Antonio C., EHRICH, Roger W.Computer Analysis of User Interface Based on Repetition in Transcripts of User Sessions. ACM Transactions on Information Systems, New York-USA, v.9, n.4, p.309-335, Oct. 1991.

[SIO291] SIOCHI, Antonio C., HIX, Deborah. A Study of Compute-Supported User Interface Evaluation using Maximal Repeating Pattern Analysis. Human Factors in Computing Systems - CHI'91, New Orleans-USA, p.301-305, Apr/May. 1991.

[SOND82] SONDHEIMER, Norman K. e RELLES, Nathan. Human Factors and User Assistance in Interaction Computing System: An Introduction, IEEE Transactions on System, USA, n.12, p.102-107, Mar/Apr. 1989.

[SWAN94A] SWAN, Tom. Programação Avançada em Borland C++4 para Windows. Tradução por Jorge Cokles costa de Oliveira. São Paulo: Berkeley, 1994. p549-578.

[SWAN94B] SWAN, Tom. Formatos de Arquivos para Windows 3.1. Tradução por Marcelo Vieira de Brito. Rio de Janeiro: Berkeley, 1994. 306p.

[WIEN91] WIENER, Rchard S., PINSON, Lewis J. Programação Orientada para Objeto e C++. Tradução por Andrea Nastri; revisão técnica por José Roberto Adorni Martins. São Paulo: Makron, McGraw-Hill, 1991. 333p.

[WINB93] WINBLAD, Ann L., EDWARDS, Samuel D., KING, David R. Software Orientado Ao Objeto. Tradução por Denise de Souza Boccia; revisão técnica por José Cláudio Boccia. São Paulo: Makron Books, 1993.

[TREU94] TREU, Siegfried. User Interface Evaluation; A Structured Approach. New York-USA: Plenum Press, 1994. 282p.

[TURN92] TURNELL, Maria de Fátima Q. V. Interface Homen-Máquina.Campina Grande-PB: UFPB/COPELE, 1992. 119p. (Notas de aulas).

[YALL91] YALLOUZ, Carlos. Programas Residentes No IBM PC. Rio de Janeiro: LivrosTécnicos e Científicos Editora Ltda, 1991. 183p.

[YAO95] YAO, Paul. Borland C++ 4.0: Programção for Windows™. Tradução por João Eduardo Nóbrega Tortello; revisão técnica e adaptação dos programas por Jeremias René D. Pereira dos Santos. São Paulo: Makron Books, 1995. 983p.

# Apêndice A

# Bibliotecas de ligação dinâmica (DLL).

Uma biblioteca é uma ferramenta conveniente para tratar com uma coleção de módulos objetos como uma simples unidade. As bibliotecas de ligação dinâmica (DLL - "*Dynamic Link Libraries*") são um dos elementos estruturais mais importantes do *Windows*. A maior parte dos arquivos em disco associados ao *Windows* são módulos do programa ou módulos da DLL.

O termo ligação dinâmica refere-se ao processo que o *Windows* usa para ligar a chamada a uma função em um módulo, à função verdadeira no módulo da biblioteca [PETZ93]. DLLs são arquivos que contêm funções e *procedures* que podem ser usadas pelos programas [NELS94].

Um módulo da DLL pode ter qualquer extensão (**.EXE, .FON**, etc.), porém só os módulos com extensão **.DLL**, (extensão padrão do *Windows*), são carregados automaticamente pelo *Windows*. Os arquivos com outras extensões precisam que o próprio programa que as chamam utilizem a função ***LoadLibrary*** para carregá-las [PETZ93].

## A.1 - Biblioteca Dinâmica X Biblioteca Estática.

Como seu nome sugere, DLLs são bibliotecas de procedimentos que uma aplicação pode ligar e usar em tempo de execução, ao invés de ligar estaticamente em tempo de compilação [MICR93]. Por outro lado a "ligação estática" ocorre durante a execução do "*Link*" para criar um arquivo executável (**.Exe**) a partir de vários módulos objetos (**.OBJ**) e arquivos de bibliotecas (**.LIB**) [PETZ93]. Isto significa que as bibliotecas estáticas são vinculadas ao arquivo executável do seu programa, que cresce proporcionalmente em tamanho.

As DLLs podem ser atualizadas independentemente da aplicação. A única desvantagem do seu uso, decorre do aumento do tempo de carga que é ligeiramente maior para o primeiro aplicativo que a carrega. As DLLs são muito semelhantes, em conceito às bibliotecas do DOS, ou mesmo a um arquivo (**.OBJ**) padrão.

## A.2 - Utilização das DLLs.

Um programa *Windows* é um arquivo executável que geralmente cria uma ou mais janelas e usa um laço de mensagens para receber informações do usuário. As DLLs em geral não são executáveis diretamente nem recebem mensagens. Elas são arquivos separados contendo funções que podem ser chamadas por programas e outras DLLs para executar determinadas tarefas [PETZ93].

Uma DLL é trazida à atividade quando outro módulo chama uma das funções contidas na biblioteca [PETZ93]. Logo, DLLs são especialmente preparadas para que os programas do *Windows* possam chamar rotinas dentro delas.

No *Windows*, o processo de um módulo que chama uma função em outro é generalizado. Ao escrever uma DLL, esta se estendendo o *Windows* ou pode-se pensar nas DLLs (incluindo aquelas que formam o *Windows*) como extensões dos programas que estão sendo codificados. O código, os dados e os recursos em um módulo da DLL são compartilhados entre todos os programas que usam o módulo.

As DLLs também podem fornecer recursos (mapas de *bits*, tabelas de *strings* e outras) para serem compartilhados por múltiplos aplicativos. As DLLs típicas incluem bibliotecas de controles personalizados, janelas-filhas especializadas e bibliotecas de função e recursos [SWAN94]. O propósito de uma DLL é dá aos programadores um local para armazenarem rotinas que podem ser chamadas por um ou mais programas [CALV94]. Logo, muitas aplicações podem compartilhar de uma simples Biblioteca.

Para utilizar os recurso e funções armazenadas em uma DLL (arquivo extensão **.DLL**), o programa precisa importar os componentes da DLL, como também deverá incluir um arquivo de definições (Seção A.7.) que identifique os elementos da DLL.

O *Windows* por si só é um conjunto de DLLs, as quais contém procedimentos que todas as aplicações usam para executar suas atividades, tais como: mostrar janelas e gráficos, gerenciar memória, etc. Esses procedimentos são algumas vezes referenciados como a API do *Windows*, ou interface de programação de aplicações [MICR93].

As três grandes bibliotecas que compõem o *Windows* são:

- ☐ *KERNEL*;
- ☐ *USER*;
- ☐ *GDI*.

A seguir será feita uma breve descrição das bibliotecas acima citadas e de sua utilização, segundo Yao em [YAO95].

☐ O módulo ***KERNEL*** é responsável pelo suporte de baixo nível para Entrada/Saída em arquivos, memória, carga de programa, ligações dinâmicas e tarefas tradicionalmente pertinente a um sistema operacional.

Os dois objetos mais importantes da ***KERNEL*** são a memória e os arquivos.

☐ O módulo ***USER*** trata da criação e do gerenciamento de objetos de interface de usuário, incluindo janelas, menus, caixa de diálogo, controles de caixa de diálogo, ponto de inserção, cursores e ícones. O *Windows* dá ao ***USER*** uma tarefa central, por exemplo: o escalonamento de aplicativos.

O objeto mais importante da interface do usuário é a janela.

☐ O módulo ***GDI*** é a interface de dispositivo gráfico. Quando os *pixels* iluminam a tela ou, a saída aparece em uma página criada por um aplicativo do *Windows*, estamos vendo o trabalho dessa DLL. Naturalmente, os controladores de dispositivos gráficos da ***GDI*** ajudam a fazer isso acontecer, mas a ***GDI*** é chamada pelos aplicativos, quando eles querem criar qualquer tipo de saída visual.

A ***GDI*** é o componente mais antigo do *Windows* e o objeto mais importante da ***GDI*** é o dispositivo de contexto [1] .

## A.3 - Vantagens da Utilização das DLLs.

☑ uma das principais vantagens na utilização das DLLs é, não necessitar da ligação de um programa executável que utiliza funções contidas em DLL, quando da alteração de qualquer função interna da DLL, havendo a necessidade de somente o módulo ser ligado. Isto denota que a biblioteca pode ser alterada independentemente da aplicação. Diferentes dos arquivos (**.OBJ**) ou (**.LIB**), as rotinas armazenadas numa DLL são ligadas em tempo de execução. Durante a Compilação um programa é meramente informado da presença de uma DLL que contém as rotinas que o programa necessita. Após o programa ser carregado, essas rotinas são ligadas dinamicamente [CALV94].

☑ Uma outra vantagem decorre da utilização, por vários aplicativos das funções embutidas nas DLLs. Elas preservam memória ao consolidar as subrotinas mais comuns em um único módulo em vez de utilizar múltiplas cópias em cada programa que as requeiram [SWAN94]. Isto significa que muitas aplicações podem compartilhar uma simples DLL, requerendo menos espaço em disco para os arquivos e menos espaço na memória ao executar duas ou mais cópias do aplicativo.

Para otimizar o uso da memória, o ***KERNEL*** mantém apenas uma cópia de um módulo presente na RAM, mesmo quando ele é acionado por vários usuários. Por exemplo, duas cópias do aplicativo "bloco de notas" só exigem que uma cópia do código e dos recursos estejam presentes na memória . O mesmo é válido para as DLLs, uma única cópia de ***KERNEL***, ***USER*** e ***GDI*** serve a todos os aplicativos *WinApi* [YAO95].

---

[1] Dispositivo de contexto (DC) é um elemento que faz a ligação (LINK) entre uma aplicação *Windows*, um " driver" de dispositivo, e um dispositivo de saída, como uma impressora.

## A.4 - Procedimentos em uma DLL.

Se for usada a palavra chave *_export* quando for definido um procedimento, todas as funções membros que não estão em linha[2] e todos os dados estáticos membros desta classe são exportados.

Quando um programa *Windows* é carregado, uma chamada ***Far*** dentro do programa aponta para a entrada de uma função dentro de uma DLL, a qual se encontra em memória. Como o código dentro de uma DLL não está no mesmo segmento onde se encontra o código do programa, a função *Windows* deve ser definida como ***Far***. Também os apontadores passados dentro da função Windows devem ser definidos como ***Far*** para evitar conflito com o próprio código e o segmento de dados da DLL.

Exemplo da utilização da palavra chave *_export* e ***Far***

A função Abrir Projeto em **Dinamico.dll**

```
int Far Pascal _export Abrir_Projeto(LPSTR);
```

Na DLL o fonte **Dinamico.cpp**, contém a implementação da função da seguinte maneira:

```
int [Far Pascal| CALLBACK][3] export Abrir_Projeto(LPSTR lpszProjeto)
{
        corpo da função
        return;
}
```

## A.5 - Construção de uma DLL

Existem dois tipos de arquivos DLL, que são:

- ☑ *Windows* API DLLs;
- ☑ DLLs Terceirizadas.

❑As *Windows* API DLLs são enviadas com o pacote *Windows*, e, desta forma os programas podem seguramente supor que possuem estas DLLs instaladas no PC em que ele será executado [GURE95].

❑ As DLLs Terceirizadas contêm funções e *procedures* que estenderão ainda mais os recursos disponíveis.

Também pode-se criar arquivos .DLL. Normalmente, a maioria dos programadores escreve DLLs usando a linguagem de programação **C.**. Pode-se usar o compilador C da

[2] Funções que não estão em linha são as funções que são chamadas dentro das funções "export".
[3] No WINDOWS.H CALLBACK é um #define de FAR PASCAL

*Microsoft*, da *Borland* ou qualquer outro. Os programadores devem se certificar que o pacote destes compiladores inclui o *SDK for Windows* [GURE95].

Segundo [PORL94], para criar um arquivo de Biblioteca de ligação dinâmica (DLL), o arquivo fonte em C precisa de apenas três partes:

❶ - Uma função ***LibMain***, que realiza qualquer inicialização necessária;

❷ - Uma *procedure* de saída do *Windows* cujo nome deve ser ***WEP*** *(Windows Exit Procedure)*. Na realidade a função ***WEP*** original já se encontra na biblioteca do *Windows*;

❸ - E, As funções chamadas externamente que compõem a própria DLL.

A seguir serão detalhados os itens que compõem as partes principais da criação de uma DLL.

❶ - A função ***LibMain***

Para as bibliotecas de ligações dinâmicas, o código de carga é fornecido no ***LibEntry.Obj*** (*Microsoft*) ou no ***Cows.Obj*** (*Borland*). O *Windows* chama ***LibEntry*** uma vez (quando o primeiro programa que requer a DLL é carregado) com os registradores do microprocessador contendo:

DI - *Handle*[4] da cópia;
DS - segmento de dados da cópia;
CX - Tamamho do *Heap*;
ES=Si - Linha de comando.

Na função ***LibMain*** não é necessário um registrador que contenha o tamanho da pilha, uma vez que os módulos da biblioteca não têm pilha. Não é necessário um registrador que contenha o "*handle*" da cópia anterior, pois os módulos DLLs não podem ter múltiplas cópias na memória. O parâmetro da linha de comando em ES=SI não é usado na maioria das DLLs. O *LibEntry* precisa retornar um valor diferente de zero (0) se a inicialização for bem sucedida e zero (0) caso contrário. Uma falha na inicialização faz com o *Windows* não execute o programa que requer a biblioteca [PETZ93].

❷ O *procedure* ***WEP***.

Este "procedure" é uma rotina de "desinicialização". Quando ***WEP*** é chamada pelo *Windows*, o parâmetro *wParam* pode ser igual a ***WEP_FREE_DLL*** para indicar que o

[4] Um "handle" é um inteiro de 16 bits sem sinal, único, que o *Windows* usa para identificar um objeto criado ou usado por uma aplicação.

último programa que usa a DLL está terminando e ela não é mais necessária, ou pode ser igual a *WEP_SYSTEM_EXIT* para indicar que o *Windows* está se encerrando. Em ambos os casos, deve-se fazer a rotina retornar 'um' (1).

Como o *Windows* procura uma função chamada *WEP* na DLL, cuidado com a desfiguração dos nomes [5] no C++. Use o nome da função no alto do programa dentro de uma declaração ***extern*** [PETZ93].

Ex.:
```
extern "C"
{
        int CALLBACK _export WEP(int wParam);
}
```

❺ - As funções chamadas externamente que compõem a própria DLL.

Estas funções são definidas pelos programadores, da mesma maneira, como são definidas as demais funções de um programa executável, levando em consideração os elementos que as tornam exportáveis ( O ***Far*** e ***_export***, que já foi visto anteriormente).

Deve-se ter um cuidado especial em relação aos nomes da funções que são exportadas (***_export***). Para evitar a sua desfiguração, deve-se proceder da mesma maneira como foi feito com a função *WEP*. A declaração ***extern*** seguida por "C", instrui o compilador a não desfigurar as funções definidas dentro das chaves. O programa de ligação ("*link*") não resolverá uma chamada de função em um módulo com a função em outro módulo se os parâmetros forem incorretos [PETZ93]. Se as funções exportadas de uma DLL têm os seus nomes desfigurados durante a fase de compilação, um programa só poderá usar a DLL se for compilado com o mesmo compilador, afetando assim a portabilidade da DLL ou do *software* que a utiliza.

Para ser construído um arquivo em C, deve-se incluir o arquivo de cabeçalho, *WINDOWS.H* e os demais cabeçalhos que se façam necessários. Isso é feito com o comando do pre-processador C.

A seguir será apresentado um pequeno trecho, a partir do qual será mostrado a construção de uma DLL, cujos procedimentos serão utilizados por um programa de coleta de dados.

```
//-------------------------------------------------------------------------
// Dinamico.h  - Arquivo de cabeçalho
//          Eliane da S. Alcoforado Diniz - 1996
//-------------------------------------------------------------------------
#define STRICT
#include <WINDOWS.H>
```

[5] Desfiguração dos nomes - quando uma aplicação é compilada no modo C++, os nomes das funções são alterados com a inclusão de códigos que indicam os parâmetros das funções e o valor de retorno. Isso permite a sobrecarga de funções do C++ e oferece uma certa conferência de erros durante a ligação

```
#include <string.h>
#include <Windowsx.h>
#pragma warning (disable:4068)
#pragma argsused
//-----Estrutura para receber os dados do Sistema-----
 typedef struct Dado_Projetos
 {
        BYTE      szSistema[61];
        BYTE      szProjeto[31];
        BYTE      szProjetista[31];
        BYTE      szSigProj[9];
        BYTE      szVersao[3];
        DWORD     nDataColeta;
 }Projetos;

//------Ponteiro para a estrutura enviada para a Função C/ dados do Projeto-----
static Projetos *lpszProjetos;

//--------------------Protótipo das funções  Exportavéis-----------------------------
extern "C"
{
 int CALLBACK _export Abrir_ProjDin(LPSTR);
 int CALLBACK _export Grava_ProjDin();
 void CALLBACK _export Cancel4_Abertura();
 void CALLBACK _export Receber_Estrut(Projetos *);
}

//-------------------------Protótipos das funções Internas----------------------------
 void Limpar_Variaveis();

//-----------------------------Variáveis Globais-------------------------------------
LPSTR lpszSigla1;              //Guarda o endereço da Sigla do Projeto-ponteiro


//--------------------Ponteiro para o arquivo-----------------------------------------
FILE *lpArqPDin = NULL;   //Arquivo que contém todas as versões e Sessões
da coleta                 //Dinâmica
```

Seguir o trecho da DLL, onde será construída as funções cujos protótipos encontram-se no **Dinamico.h**

```
//---------------------------------------------------------------------------
//Dinamico.cpp - módulo de biblioteca para o programa de Coleta Dinâmica
//          Eliane Alcoforado Diniz - 1996
//---------------------------------------------------------------------------
#include "dinamico.h"
//Protótipo das funções WEP()
extern "C"
{
  int CALLBACK WEP (int nparameter);
}
//---Função Principal da Biblioteca - LibMain---
//---Nesta função é realizada a inicialização para o restante das funções na DLL pois  é ---
//---chamada, quando a biblioteca é carregada inicialmente.---
#pragma argsused
int CALLBACK LibMain(HINSTANCE hInstance, WORD wDateSeg,
  WORD wHeapSize, LPSTR lpszCmdLine)
{
  if(wHeapSize > 0)
  UnlockData(0);
  return 1;
}
//-----Função WEP  é a procedure de saída do Windows-----
#pragma argsused
int CALLBACK WEP (int nparameter)
{
  return 1;
}


//------------------------------ As Funções Exportavéis----------------------------
//-----Abrir arq. de Projeto da Coleta Dinamica------
//-----lpszNomeProj aponta para a Sigla do Projeto Dinâmico-----
//-----Retorna:   1 se erro na Abertura
//                0 se Abertura Ok
```

```
#pragma argsused
int CALLBACK _export Abrir_ProjDin(LPSTR lpszNomeProj)
{
  lpArqPDin=NULL;
  lpszSigla1=lpszNomeProj;
  strcpy(szNomePDin," ");
  strcpy(szNomePDin,"f:\\eliane\\visual\\");
  strcat(szNomePDin,lpszNomeProj);
  strcat(szNomePDin,".pjd");
  if((lpArqPDin=fopen(szNomePDin,"at+"))==NULL)
        return 1;
  return 0;
}
```

//--Rotina p/ Receber a Estrutura , com dados do sistema, utilizada na coleta dinâmica--

```
#pragma argsused
void CALLBACK _export Receber_Estrut(Projetos *lpszEstrutSis)
{
  lpszProjetos=lpszEstrutSis;
}
```

//---Rotina p/ fechamento do Arq. de Projeto, Quando da Opção de Cancelamento---

```
#pragma argsused
void CALLBACK _export Cancel4_Abertura()
{
  if (lpArqPDin!=NULL)          Arquivo de  projeto Dinâmico
        fclose(lpArqPDin);
}
//----- Rotina que grava o Arq. de Projeto da Coleta Dinâmica-----
#pragma argsused
int CALLBACK _export Grava_ProjDin()
{
  DWORD POSICAO=0L;
  if(fseek(lpArqPDin,POSICAO,2)!=0)
        return 1;
```

```
  if(fwrite(lpszProjetos,sizeof(Projetos),1,lpArqPDin)!=1)
        return 2;
  POSICAO=ftell(lpArqPDin);
  if(fseek(lpArqPDin,POSICAO,0)!=0)
        return 1;
  if(fwrite(lpszPerfilUsuario,sizeof(Perfil),1,lpArqPDin)!=1)
        return 3;
  POSICAO=ftell(lpArqPDin);
  if(fseek(lpArqPDin,POSICAO,0)!=0)
        return 1;
  if(fwrite(szNomeDin,sizeof(szNomeDin),1,lpArqPDin)!=1)
        return 4;
  return 0;
}

//------------ Rotinas Internas da DLL ----------
//Rotina para limpar todas as variaveis quando sair da DLL
#pragma argsused
void Limpar_Variaveis()
{
  lpszSigla1 = NULL;
  strcpy(szNomePDin," ");
}
```

## A.6 - Conectando as funções de uma DLL a um programa em Visual Basic.

Os procedimentos DLL, residem em arquivos que são externos à aplicação *Visual Basic* (VB). Logo, pode-se usar DLL com o VB, porém não é possível escrevê-las usando o VB. Decorrente disto, os programadores podem tirar vantagem em usar as funções da *API* do *Windows* que realizam tarefas que não estão disponíveis no Visual *Basic*. Quando se desejar usar um procedimento externo ao programa são necessários dois passos:

❶. Declarar o procedimento externo;

❷. Fazer a chamada ao procedimento.

❶. Declarar o procedimento externo;

Para se usar uma função que se encontra em uma DLL nos programa VB, é necessário declará-la. O uso da palavra chave ***Declare*** informa ao VB, onde encontrar

os procedimentos DLL que se deseja utilizar. A declaração ***Declare*** deve ser colocada dentro de uma seção de declaração geral de um formulário. Neste caso, ela só poderá ser usada pelo formulário no qual foi declarada. Poderá também ser declarada em uma seção de declaração de módulo de código (.BAS). Neste caso ela será pública e poderá ser chamada de qualquer lugar dentro da aplicação [MICR93], ou seja, ela poderá ser chamada por qualquer formulário ou módulo de código.

As DLLS são muito parecidas com *procedures* e funções que são escritas no VB, então, se um procedimento não retorna um valor, deve ser declarado como ***Sub.***

Exemplo de declaração de Subrotinas:

❒ ***Declare Sub Receber_Estrut Lib*** **"f:\eliane\tese\estatico.dll"** (*Valor* ***As Dado_Projeto***)

❒ ***Declare Sub*** **Cancel4_Abertura** ***Lib*** **"f:\eliane\tese\dinnovo.dll"** ()

Se um procedimento retorna um valor, deve ser declarado como ***Function.***

Exemplo de Funções:

❒ ***Declare Function*** **Ler_Projeto** ***Lib*** **"f:\eliane\tese\estatico.dll"** () ***As Integer***

❒ ***Declare Function*** **Abrir_ProjDin** ***Lib*** **"f:\eliane\tese\dinamico.dll"** (***ByVal*** *SigProjeto* ***As String***) ***As Integer***

Além da palavra chave ***Declare*** e do tipo do procedimento (***Sub*** ou ***Function***), que compõem a seção de declaração deve-se ressaltar os demais componentes:

No exemplo, ***Declare Function*** **Abrir_ProjDin** ***Lib*** **"f:\eliane\tese\dinamico.dll"** (***ByVal*** *SigProjeto* ***As String***) ***As Integer***

**Abrir_ProjDin** - Nome do procedimento dentro da DLL

***Lib*** - Palavra chave indicando que o elemento seguinte é o local onde se encontra o procedimento.

**"f:\eliane\tese\dinamico.dll"** - especificação do local. Pode como no exemplo acima, incluir o caminho (*Path*), ou simplesmente o nome da DLL (**dinamico.dll**).

(***ByVal*** *SigProjeto* ***As String***) - Passagem de parâmetro, caso seja necessário.

***As Integer*** - Como se trata de uma função, este elemento da declaração indica o tipo de dado que será retornado.

Na declaração deverá ser tomado um cuidado especial em relação à passagem de parâmetros, já que o VB não verifica a passagem correta. Caso o nome da Biblioteca ou os parâmetros estejam incorretos, haverá um *crash*, devendo a aplicação ser relocada e restaurada, perdendo todas as alterações efetuadas após o último salvamento.

Passagem de Parâmetros.

No *Visual Basic* existem duas maneiras de passar os parâmetros para os procedimentos, que são:

⊙ - Passagem por Referência;

⊙ - Passagem por Valor.

⊙ - Passagem por Referência.

A passagem de parâmetros no VB, por *default* é por referência, o VB fornece um endereço do tipo ***Far*** com comprimento de 32 bits [MICR93]. Neste caso qualquer alteração na variável dentro do procedimento afetará o valor inicial da variável, que foi passada como argumento.

⊙ - Passagem por Valor - Para passar um argumento por valor coloca-se a palavra chave ***ByVal*** na frente do argumento dentro da seção de declaração. Isto indica que toda vez que um argumento for passado por valor, o seu valor inicial não será alterado e, uma cópia será gerada dentro do procedimento chamado.

Alguns procedimentos em DLL aceitam mais de um tipo de dado para o mesmo argumento, neste caso, deve-se na declaração do procedimento, passar o argumento com ***As Any*** para remover as restrições do tipo.

Exemplo: ***Declare Sub*** **Nome do Procedimento** ***Lib*** **"nome da DLL"** (***ByVal*** *variável1* ***As Integer***, *variável2* ***As Any***)

Quando este procedimento é chamado, pode-se passar o último argumento, tanto como um longo, quanto como uma *string*.

Ex.: Da passagem do argumento como um longo:

```
Sub ....
        Dim Receber As Long
        Dim Valor As Long
        Dim VarCont As Integer
        Valor = ???
        Receber=Nome do Procedimento(VarCont,ByVal Valor)
End Sub
```

Ex.: Da passagem do argumento como uma *String*:

```
Sub ....
        Dim Receber As Long
        Dim Val As String
        Dim V      as Integer
```

*Val*=txtform.text

*Receber*=**Nome do Procedimento**(*VarCont*,*ByVal Val*)

***End Sub***

A palavra chave ***ByVal*** - é usada para fazer restrições aos tipos de dados. Quando ela é utilizada, o VB entende que a passagem é por valor, eliminando a passagem *default*. No entanto quando o argumento é uma *string*, ela tem uma outra conotação, ela converte uma string do VB para uma *String* com terminação nula, como é utilizado na linguagem C, por exemplo.

Passando um *Array* como Argumento.

Conforme [MICR93] pode-se passar elementos individuais de um *array* do mesmo modo que é passado qualquer variável. Por exemplo, pode ser usado o procedimento *sndPlaySound* (declarado facilmente) para tocar uma série de arquivos *.WAV* armazenados em um *array*:

```
Dim i As Integer, worked As Integer
for i=0 to UBound(WaveFiles)
        worked = sndPlaysound(waveFiles(i),0)
next i
```

Passando um Tipo de Dado definido pelo Usuário

Alguns procedimentos em DLL usam como argumento um tipo de dado definido pelo usuário. O tipo de dado definido pelo usuário não pode ser passado por valor, eles têm que ser passados por referência. Estes tipos de dados são conhecidos na linguagem C por estrutura (*Struct*). O VB passa o endereço do primeiro elemento e os elementos restantes são colocados na memória seguindo o primeiro elemento. A seguir será mostrada uma estrutura e a chamada a um procedimento que a utiliza:

'Estrutura para guardar os dados do Projeto

```
Type Dado_Projeto
  szNomesist As String * nTamString1
  szNomeProj As String * nTamString2
  szNomeprojet As String * nTamString2
  szSigProj As String * nTamString3
  szVersao As String * nTamString4
  nDataColeta As String * nTamString3
End Type
```

*Global Projeto* As Dado Projeto 'A variável Projeto reserva uma área de memória

*Declare Sub* **Receber_Estrut** *Lib* **"f:\eliane\tese\estnovo.dll"** (*Valor As* Dado Projeto)

Agora possível utilizar o seguinte procedimento para passar a estrutura para a DLL

```
Sub cmd3faiw8_Click ()
  Dim nRetorno As Integer
  If txt12faiw8 = "" Then
  txt12faiw8.SetFocus
  Else
  txt14faiw8 = ""
  Limpar_Estrutura
  Projeto.szNomesist = txt6faiw8.Text
  Projeto.szNomeProj = txt8faiw8.Text
  Projeto.szNomeprojet = txt10faiw8.Text
  Projeto.szSigProj = txt2faiw8.Text
  Projeto.szVersao = txt4faiw8.Text
  Projeto.nDataColeta = txt12faiw8.Text
  Receber_Estrut Projeto
End Sub
```

A DLL que contém este procedimento encontra-se descrita, anteriormente na seção A.5.

❷. Fazer a chamada ao procedimento.

No *Visual Basic* a declaração de uma função ou de uma subrotina é efetuada uma única vez, no entanto esta função ou esta subrotina poderá ser chamada tantas vezes quantas sejam necessárias.

Uma vez que a declaração do procedimento e a passagem de parâmetros tenham sido efetuadas de maneira correta, é possível fazer a chamada ao referido procedimento, simplesmente fazendo a chamada como se procede com qualquer outro procedimento em sua aplicação.

Dentro de um programa *Visual Basic* quando é necessária uma chamada a uma dada DLL deve-se proceder da seguinte maneira:

① Exemplo de um procedimento que fecha os arquivos que estiverem abertos:

**Cancel4_Abertura** - Esta subrotina tem a função de fechar todos os arquivos que estiverem abertos quando houver alguns erro no processamento ou na finalização da opção

② Exemplo da leitura dos dados pertencentes a um dado projeto:

*nRetorno* = **Ler_Projeto**() - Esta função não necessita passar nenhum argumento, ela retorna um inteiro, para posterior verificação da sua ocorrência

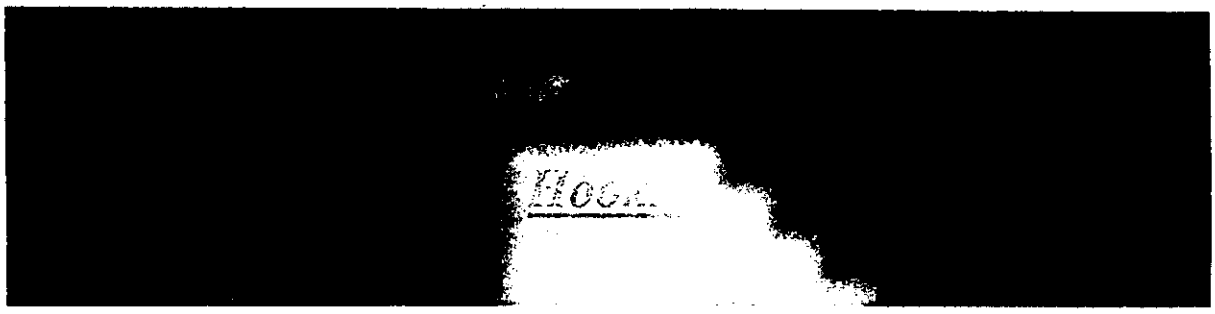
## A.7 O arquivo de definições - .DEF

Segundo [CALV94], para compilar uma DLL, devem ser feitas definições diferentes das que podem ser utilizadas para as partes normais dos programa. Se estiver utilizando as IDES da *Borland* ou *Microsoft*, pode-se simplesmente utilizar os menus fornecidos para obter automaticamente essas definições, conforme descrito na sua documentação. Se estiver utilizando um *makefile*, deve-se utilizar o arquivo *.DEF*, com as seguintes definições:

- Utilizar o comando *LIBRARY* ao invés de *NAME*;
- Declarar como único e não como múltiplo;
- Não possuir o comando *STUB*;
- Não possuir o comando *STACKSIZE*;

A seguir apresentamos o arquivo *.DEF* para ser executado com o *Windows*:

*DESCRIPTION* Mostrando o *.DEF* para uma DLL.'
*EXETYPE WINDOWS*
*CODE PRELOAD MOVEABLE DISCARDABLE*
*DATA PRELOAD MOVEABLE SINGLE*
*HEAPSIZE* 5120
*EXPORTS* 1ª FUNÇÃO
2ª FUNÇÃO
. . .
Nª FUNÇÃO



Um *hook* é uma sub-rotina instalada no mecanismo de tratamento de mensagem do *Windows*, que permite monitorar e capturar certos tipos de mensagens [YAO95]. Logo, um *hook* é um ponto, dentro do mecanismo de manipulação de mensagens do *Windows*, que uma aplicação pode usar para ter acesso ao laço de mensagens. Ele dá a uma aplicação um meio de poder laçar um evento que esteja ocorrendo em uma dada aplicação ou no sistema *Windows* [RICH92].

Todos os *hooks* que são instalados em nível de sistema, afetarão todas as aplicações que estão em execução no sistema [YAO95]. Segundo [RICH92], toda vez que um evento ocorre no sistema, o *Windows* tem que notificar todos os *hooks* que estão instalados para aquele evento. Isto pode notavelmente degradar o desempenho do *Windows*. O *Windows* 3.1 tenta solucionar este problema permitindo que, a aplicação instale um *hook* que será notificado somente dos eventos que sejam relevantes para a aplicação.

## B.1- Tipos de Hooks e a sua utilização.

O *Windows* fornece vários tipos de *hooks*; cada um provê acesso a um tipo particular ou classe de mensagem.

A partir da versão 3.1 do *Windows*, 12 tipos de *hooks* foram definidos na "*WinAPI*". (Tab B.1).

| Tipos | Descrição |
|---|---|
| *WH_CALLWNDPRC* | Este tipo é chamado quando um procedimento de janela está para receber uma mensagem não enfileirada. |
| *WH_CBT* | Este *hook* recebe notificação, quando uma janela é ativada, criada destruída, minimizada, recebe o foco ou recebe uma mensagem de comando do sistema. |
| *WH_DEBUG* | Este *hook* é chamado antes de todos os demais. Ele ajuda a depurar código de *hook* |
| *WH_GETMESSAGE* | Este *hook* é ativado imediatamente antes que uma mensagem enfileirada seja recuperada por *GetMessage/PeekMessage*. |
| *WH_HARDWARE* | É usado para dispositivos de *hardware* fora do padrão. |
| *WH_JOURNALPLAYBACK* | Reproduz eventos de teclado e de *mouse* gravados previamente. |
| *WH_JOURNALRECORD* | Grava eventos de teclado e de *mouse* para reprodução posterior. |
| *WH_KEYBOARD* | Faz notificação sobre entrada de teclado. |
| *WH_MOUSE* | Faz notificação sobre entrada de *mouse*. |
| *WH_MSGFILTER* | Faz a notificação de laço de mensagem de menus e diálogos. |
| *WH_SHELL* | Faz a notificação sobre a criação e destruição de janelas de aplicativos de nível superior. |
| *WH_SYSMSGFILTER* | Faz a notificação em nível de sistema do laço de mensagem de menus e diálogos. |

**Tab B.1.Relaciona 12 tipos de *Hooks* e faz uma breve descrição da sua funcionalidade.**

Após terem sidos relacionados os tipos de *hooks* existentes na versão Windows 3.1. passamos a descrever apenas os tipos de *hooks* a serem utilizados no projeto de avaliação de interfaces discutido neste trabalho, que aqui é utilizado como estudo de caso.

- ***Hook* de Teclado -*WH_KEYBOARD*.**

Todo programa *Windows* tem um laço ***GetMessage*** ( ou ***PeekMessage***) responsável por introduzir todas as mensagens de teclado em um programa. Quando os dados brutos do teclado chegam em um programa *Windows*, uma rotina de biblioteca especial (***TranslateMessage***) deve ser chamada, com a finalidade de manipular os dados para criar a entrada de caracteres realmente útil.

Sempre que pressionamos ou liberamos uma tecla, o *hardware* do teclado gera um código de varredura (CV) de um ou dois *bytes*, que identifica a tecla. Toda tecla produz dois CV um ao ser pressionada outro ao ser liberada.

Devido a internacionalização do teclado, o controlador do teclado só precisa do CV correto para o código de tecla e deste para as tabelas de conversão *ASCII*. A Tabela B.2 lista algumas teclas do teclado virtual, junto com seus nomes simbólicos. A lista completa encontra-se no *WINDOWS.H*.

| (Hexa) | (Dec) | Nome Simbólico | Tecla Pressionada |
|---|---|---|---|
| 1 | 1 | *VK_LBUTTON* | |
| 3 | 3 | *VK_CANCEL* | *Ctrl+Break* |
| 8 | 8 | *VK_BACK* | *Backspace* |
| D | 13 | *VK_RETURN* | *Enter* |
| 10 | 16 | *VKSHIFT* | *Shift* |
| 1B | 27 | *VK_ESCAPE* | *Esc* |
| 2F | 47 | *VK_HELP* | <Sem uso> |
| 71 | 112 | *VK_F1* | Tecla de função F1 |

**Tab B.2 Código de teclas virtuais. continuação**

Quando as teclas são pressionadas ou soltas, os CVs são enviados para o circuito de controle da placa do sistema do computador. Quando o circuito de controle percebe que a entrada de teclado está disponível, ele gera uma interrupção de *hardware* 09H. Quando se esta trabalhando com o DOS, esta interrupção resulta em uma chamada à rotina de tratamento de teclado do BIOS. Mas, quando o *Windows* está presente, outro mecanismo é utilizado, o qual satisfaz os requisitos especiais que o *Windows* tem para o tratamento da entrada de teclado. Este tratamento é feito pela rotina de tratamento de interrupção - *Hook* do Teclado.

Este *hook* infiltra-se no fluxo de mensagens de teclado vindo da fila de eventos do *hardware* e oferece um meio de monitorar todas as entradas do teclado do sistema (Fig. B.1). Logo, criar um *hook* de teclado no *Windows* é comparável a roubar a interrupção de teclado do DOS: ele fica com o controle total sobre o fluxo de mensagens de digitação no sistema.

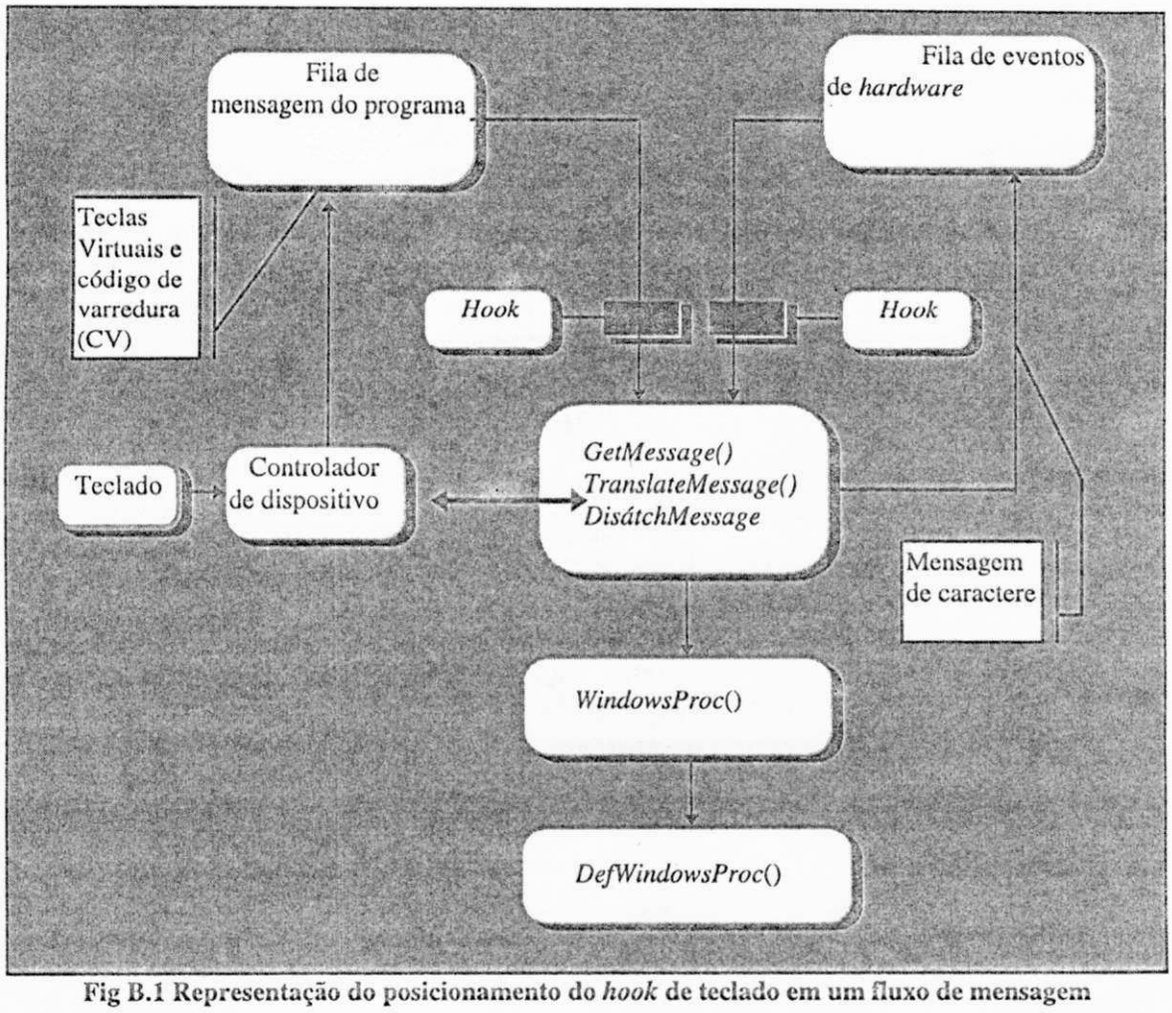



**Fig B.1 Representação do posicionamento do *hook* de teclado em um fluxo de mensagem**

Este tipo de *hook* é usado por uma aplicação quando se quer examinar o valor enviado pelo teclado ao pressionar ou liberar uma tecla, e a aplicação não possui o foco de entrada.

Quando o usuário pressiona ou libera uma tecla, a mensagem gerada por esse evento é colocada pelo *Windows*, dentro da fila do sistema do *Windows*, o qual posteriormente a transfere para a fila de mensagem da aplicação. Esta mensagem é retirada da fila, quando uma aplicação faz uma chamada a ***GetMessage*** ou ***PeekMessage***. Antes do *Windows* liberar o evento para o laço de mensagens da aplicação, ele chama a primeira função filtro dentro da cadeia .

Este tipo de *Hook* deverá estar em uma Biblioteca de Ligação dinâmica (DLL).

**Parâmetros do *Hook* de teclado.**

A Tabela B.3, relaciona os valores passados na função filtro , *LRESULT CALLBACK* ***KEYBOARDProc****(nCode, wParam, lParam)*, e o valor retornado, esperado pelo *Windows*:

| *Cod* | *wParam* | *lParam* | *lResult* |
|---|---|---|---|
| *HC_ACTION* | Esp. o cód. virtual da tecla | Esp. a mesma informação que é enviada dentro do parâmetro *lParam* para um procedimento de janela quando ela recebe a mensagem *"WM_ KEYDOWN"*. | Este resultado é zero se a mensagem for processada ou um se a mensagem for descartada. |
| *HC_NORE-MOVE* | Esp. o cód. virtual da tecla | Esp. a mesma informação que é enviada dentro do parâmetro lParam para um procedimento de janela quando ela recebe a mensagem *WM_ KEYDOWN*. | Este resultado é zero se a mensagem for processada ou um se a mensagem for descartada. |

**Tab B.3 Relaciona os valores contidos nos Parâmetros da função filtros do *hook* de teclado.**

Cod - Especifica se a função filtro poderá processar a mensagem ou chamar a função ***CallNextHookEx***. Se este valor for *"HC_NOREMOVE"*, a aplicação está usando a função ***PeekMessage*** com a opção *"PM_NOREMOVE"*, e a mensagem não será removida da fila do sistema. Se o valor for menor do que zero (0), a função filtro poderá passar a mensagem para ***CallNextHookEx*** , sem executar os procedimentos contidos na função filtro.

No *Windows*, todas as entradas de um programa, incluindo as de teclado, são enviadas na forma de mensagens, mas o significado real das mensagens de teclado vem dos valores armazenados em um valor inteiro de dois *bytes* e um valor inteiro de quatro *bytes* que compõem os parâmetros *wParam* e *lParam* de um procedimento de janela.

*wParam* - contém o código virtual da tecla que foi pressionada ou liberada. O código de tecla virtual representa como o controlador vê um evento de teclado no contexto do *Windows*.

*lParam* - é um longo dividido em seis campos (Fig B.2).

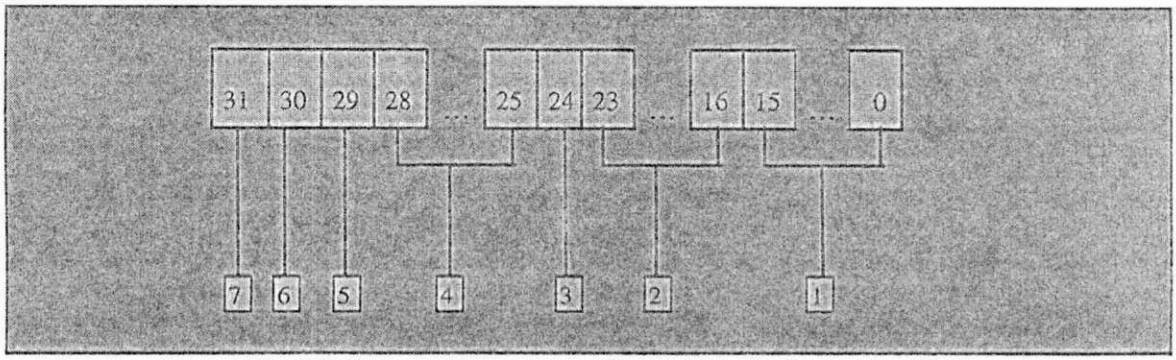



**Fig B.2 Campos da mensagem de combinação da variável *lParam***

| Campos | Descrição |
|---|---|
| ① - Número de repetição | No *hardware* do teclado existe o recurso de repetir um caracter automaticamente, caso uma tecla seja mantida pressionada. Um contador de repetição maior de que 'um' significa que os eventos de teclado estão correndo mais rapidamente do que o programa é capaz de processá-los. |
| ② - Código de Varredura (CV) | Este campo especifica o CV, que foi enviado do teclado. Para a maioria dos programas, o código de tecla virtual é mais útil, pois consiste de um código independente de dispositivo para um toque de tecla, já que o CV depende do *hardware*. Mas algumas vezes, é necessário o CV, por exemplo, quando um programa que quer diferenciar um *Shift* direito de um *Shift* Esquerdo. |
| ③ - Sinalizador de expansão | Este campo é uma extensão do CV. Ele informa a um programa que a tecla pressionada é uma das teclas duplicadas do teclado estendido da IBM. |
| ④ - Não usada | |
| ⑤ - Código de contexto | Este sinalizador tem valor 'um' (1), se a tecla *Alt* estiver pressionada; caso contrário o valor será zero (0). |
| ⑥ - Estado anterior | Este sinalizador ajuda a identificar as mensagem geradas pela ação de repetição de tecla. Ele tem o valor 'um' (1), se o estado anterior da tecla foi pressionado; e valor zero (0), se o estado anterior da tecla foi liberado. |
| ⑥ - Estado anterior | Este sinalizador ajuda a identificar as mensagem geradas pela ação de repetição de tecla. Ele tem o valor 'um' (1), se o estado anterior tecla foi pressionado; e valor zero (0), se o estado anterior da tecla foi liberado. |
| ⑦ - Estado de transição | Este sinalizador é 'um' (1), se a tecla estiver sendo liberada; e zero (0), se estiver sendo pressionada. |

**Tab. B.4. Descrição dos campos da mensagem de combinação da variável *lParam***

O Valor retornado na variável *lResult* será zero (0) se a mensagem estiver sendo processada pelo sistema e 'um' (1) se a mensagem estiver sendo descartada.

O *Windows* não permite a uma função filtro de teclado trocar os valores dentro dos parâmetros *wParam* e *lParam*, antes de enviar a mensagem para a aplicação.

• *Hook* de *Mouse* - *WH_MOUSE.*

Quando o *mouse* reporta sua posição, ele faz em termos da movimentação ao longo dos eixos x e y, e assim as ações do botão do *mouse* são reportadas como pressionamento e liberação. Quando ocorre um desses eventos, é enviado um sinal que resulta na geração de uma interrupção de *hardware*.

Como os eventos de *hardware*, os eventos de *mouse* não são enviados aos programas durante a interrupção. Em vez disso, eles são colocados na fila de eventos de *hardware* do *Windows*.

Todo programa *Windows* tem um laço de mensagem. Uma aplicação obtém o valor da mensagem através de uma chamada a ***GetMessage***, que serve como uma das passagens para as mensagens entrarem em um programa para o processamento.

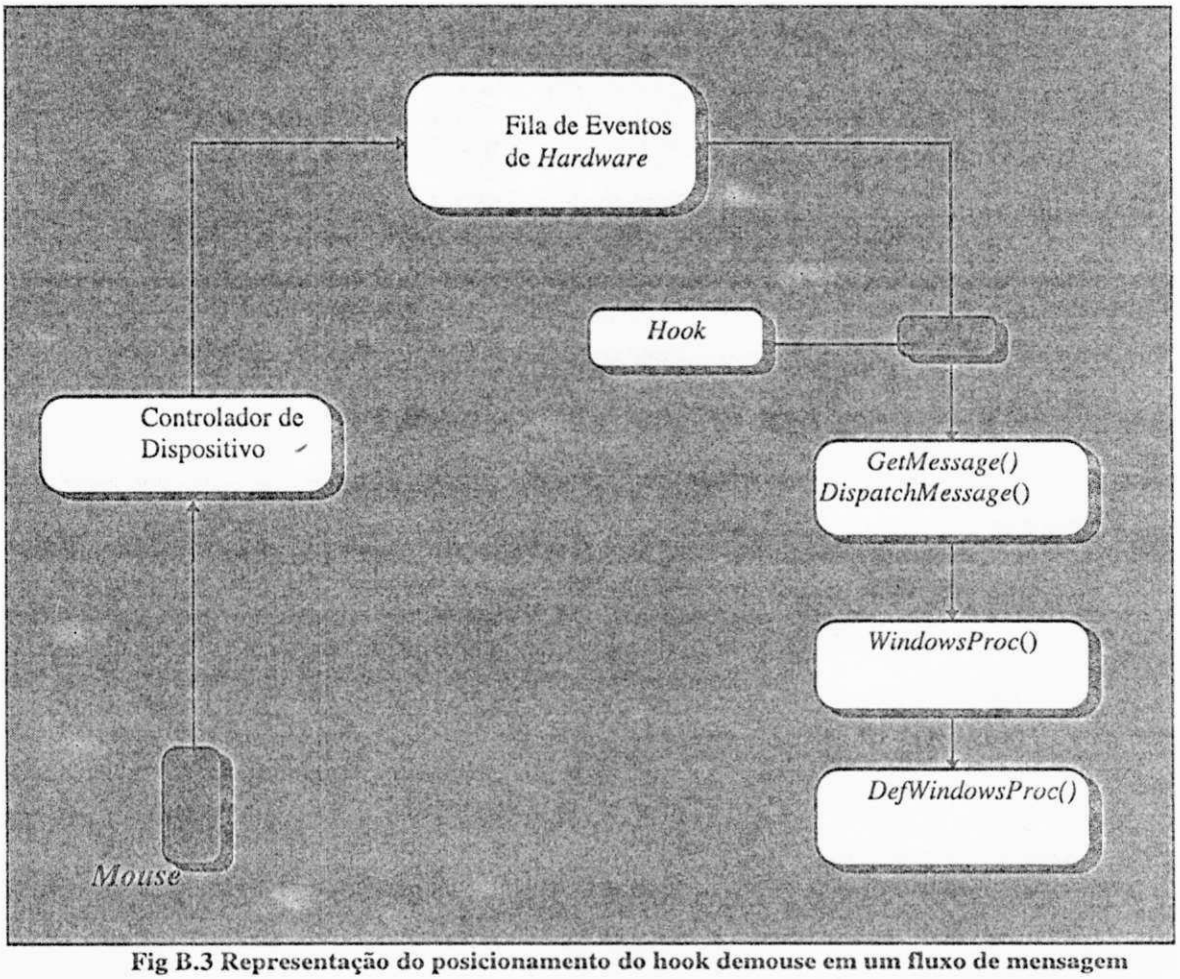



**Fig B.3 Representação do posicionamento do hook demouse em um fluxo de mensagem**

Quando *GetMessage* está pronta para trazer a mensagem para a aplicação, faz uma chamada ao I do *mouse* para ver se alguma mudança precisa ser feita, antes que a mensagem seja entregue ao programa (Fig. B.3).

Este *hook* é usado pela aplicação quando se quer examinar os movimentos do mouse e os "cliques" dos botões. Neste caso a aplicação que instala um *hook* de *mouse* não tem seu controle.

Quando o usuário pressiona ou libera um dos botões ou move o *mouse*, este evento é colocado dentro da fila do sistema do *Windows*. Este evento é retirado quando uma aplicação faz uma chamada a ***GetMessage*** ou ***PeekMessage***. Antes do *Windows* colocar o evento *mouse* na fila de mensagem da aplicação, ele chama a primeira função filtro, *LRESULT CALLBACK* ***MOUSEProc***(*nCode, wParam, lParam*), dentro da cadeia (Fig B.5).

**Parâmetros do *Hook* de *Mouse*.**

Para este tipo de *hook* a Tab. B.5, relaciona os valores passados na função filtro e o valor retornado:

| *Cod* | *wParam* | *lParam* | lResult |
|---|---|---|---|
| *HC_ ACTION* | Esp. mensagem do *mouse* | aponta para uma estrutura *MOUSEHOOK-STRUCT* | Este resultado é zero se a mensagem for processada ou 'um' se a mensagem for descartada. |
| *HC_NO REMOVE* | Esp. mensagem do *mouse* | aponta para uma estrutura *MOUSEHOOK-STRUCT*. | Este resultado é zero se a mensagem for processada ou um se a mensagem for descartada. |

**Tab. B.5 Relaciona os valores contidos nos Parâmetros da função filtros do *hook* de *Mouse*.**

*Cod* - Especifica se a função filtro poderá processar a mensagem ou chamar a função *CallNextHookEx*. Se este valor for *HC_NOREMOVE*, a aplicação está usando a função ***PeekMessage*** com a opção *PM_NOREMOVE*, e a mensagem não será removida da fila do sistema. Se o valor for menor do que zero (0), a função filtro poderá passar a mensagem para ***CallNextHookEx***, sem executar o processamento contido na função filtro.

*wParam* - Contém sinalizadores que descrevem o estado dos botões do *mouse* e das teclas *Shift* e *Ctrl*. O valor de um campo é 'um' (1), se o botão do mouse ou a tecla correspondente estiver pressionado; caso contrário será zero (0).

A Fig B.4 mostra os cinco campos do parâmetro *wParam* para mensagens do *mouse* na área cliente

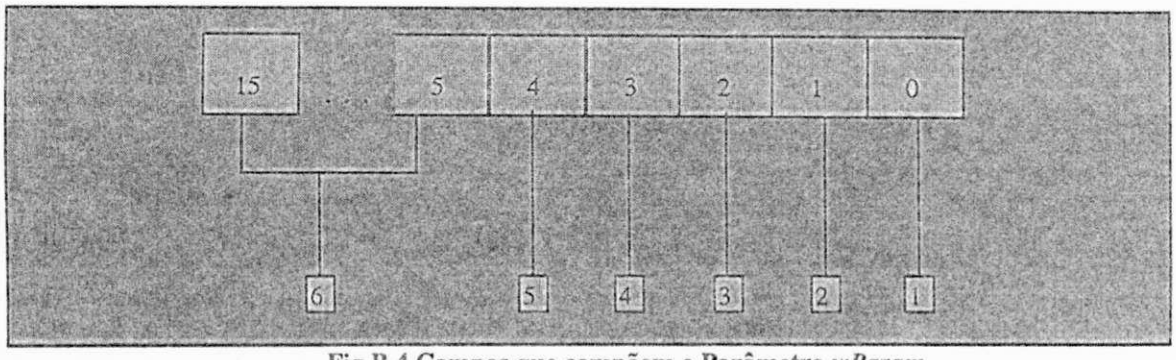



**Fig B.4 Campos que compõem o Parâmetro *wParam***

A seguir a Tab B.6 , faz uma descrição dos campos da Fig B.4

| Campos | Descrição |
|---|---|
| ① - *MK_LBUTTON* | Este campo refere-se ao botão esquerdo do *mouse*. |
| ② - *MK_RBUTTON* | Este campo refere-se ao botão direito do *mouse*. |
| ③ - *MK_SHIFT* | Este campo indica se a tecla *Shift* está ou não pressionada. |
| ④ - *MK_CONTROL* | Este campo indica se a tecla *Ctrl* está ou não pressionada. |
| ⑤ - *MK_MBUTTON* | Este campo refere-se ao botão central do *mouse*. |
| ⑥ - Sem uso | |

**Tab B.6 Detalha os campos que compõem o parâmetro *wParam* (Fig B.4.**

*lParam* - Este parâmetro contém a posição do cursor, em coordenadas da área do cliente e aponta para a estrutura ***MOUSEHOOKSTRUCT***, que contém as informações sobre o *mouse*.

A estrutura ***MOUSEHOOKSTRUCT:***

```
typedef struct tagMOUSEHOOKSTRUCT
{
        POINT pt;
        HWND hwnd;
        UINT        wHitTestCode;
        DWORD       dwExtraInfo;
} MOUSEHOOKSTRUCT;
```

A Tab. B.7, abaixo contêm informações sobre os membros da estrutura: *MOUSEHOOKSTRUCT*

| Membro | Descrição |
|---|---|
| pt | Especifica uma estrutura do tipo ***POINT*** que contém as coordenadas *x* e *y* do cursor do *mouse*, dentro das coordenadas da tela. Ainda neste relatório mostraremos esta estrutura. |
| Hwnd | Identifica a janela que receberá a mensagem que corresponde ao evento do *mouse*. |
| WhitTestCode; | Este código especifica o código *hit_test*, o qual identifica sobre qual elemento da tela o *mouse* se encontra. |
| DwExtraInfo | Especifica uma informação extra associada com o evento do mouse. Uma aplicação poderá retirar esta informação extra chamando a função de evento de hardware, *GetMessageExtrinfo*. |

**Tab B.7. Detalha os parâmetros que compõem a estrutura *MOUSEHOOKSTRUCT*.**

A seguir será mostrada a estrutura ***POINT***:

```
typedef struct tagPOINT
{
        int x;
        int y;
} POINT;
```

A Tab. B.8, mostra os membros desta estrutura:

| Membro | |
|---|---|
| x | Especifica a coordenada x de um ponto. |
| y | Especifica a coordenada y de um ponto. |

**Tab. B.8 Detalha os membros da estrutura *POINT*.**

O Valor retornado na variável *lResult* será zero (0) se a mensagem estiver sendo processada pelo sistema; 'um' (1) se a mensagem estiver sendo descartada.

- ***Hook* de *Hardware* - *WH_HARDWARE*.**

Este *hook* é usado pela aplicação quando se deseja examinar eventos de *hardware*, que não sejam oriundos do teclado nem do *mouse*. Este tipo de *hook* é usado na maioria das vezes para aplicações que rodam sob o *Windows* com a extensão ***.pen***.

Quando um evento de *hardware* ocorre, este evento é colocado na fila do sistema *Windows*. O evento é retirado quando uma aplicação faz uma chamada a ***GetMessage*** ou ***PeekMessage***.

Antes do *Windows* liberar o evento de *hardware* (exceto eventos de *mouse* ou de teclado) para a fila da aplicação ele chama a primeira função filtro, ***LRESULT CALLBACK HARDWAREProc***(*nCode, wParam, lParam*), dentro da cadeia. A Tab B.9, sumariza os valores esperados pela função filtro e o valor de retorno esperado pelo *Windows:*

| *Cod* | *wParam* | *lParam* | lResult |
|---|---|---|---|
| *"HC_ACTION"* | *NULL* | Aponta para a estrutura ***HARDWARE-HOOKSTRUCT.*** | Este resultado é zero se a mensagem for processada ou 'um' se a mensagem for descartada. |
| *"HC_NORE-MOVE"* | *NULL* | Aponta para a estrutura ***HARDWARE-HOOKSTRUCT.*** | Este resultado é zero se a mensagem for processada ou 'um' se a mensagem for descartada. |

**Tab B.9 Relaciona os valores contidos nos Parâmetros da função filtros do *hook* de *Hardware*.**

*Cod* - Especifica se a função filtro poderá processar a mensagem ou chamar a função ***CallNextHookEx***. Se este valor for ***HC_NOREMOVE***, a aplicação está usando a função *PeekMessage* com a opção ***PM_NOREMOVE***, e a mensagem não será removida da fila do sistema. Se o valor for menor do que zero (0), a função filtro poderá passar a mensagem para ***CallNextHookEx***, sem executar o processamento contido na função filtro.

*wParam* - Especifica um valor *NULL*.

*lParam* - Este parâmetro aponta para a estrutura ***HARDWAREHOOKSTRUCT***, que contém as informações sobre o *Hardware.*

A estrutura ***HARDWAREHOOKSTRUCT***:

```
typedef struct tagHARDWAREHOOKSTRUCT
{
        HWND        hwnd;
        UINT        wMessage;
        WPARAM      wParam;
        LPARAM      lParam;
} HARDWAREHOOKSTRUCT;
```

A Tab. B.10, abaixo contêm informações sobre os membros da estrutura ***HARDWAREHOOKSTRUCT:***

| | |
|---|---|
| *hwnd* | Identifica a janela que recebeu a mensagem que corresponde ao evento do *hardware*. |
| *Wmessage* | Especifica a mensagem que foi identificada. |
| *Wparam* | Especifica informações adicionais sobre a mensagem. O significado exato depende do membro *wMessage*. |
| *Lparam* | Especifica informações adicionais sobre a mensagem. O significado exato depende do membro *wMessage*. |

**Tab B.10 Detalha os membros da estrutura *HARDWAREHOOKSTRUCT***

O Valor retornado na variável *lResult* será zero (0) se a mensagem estiver sendo processada pelo sistema; 'um' (1) se a mensagem estiver sendo descartada.

***•Hook WH_CBT.***

Quando este *hook* é instalado, o *Windows* chama a função *hook* associada antes da ativação , criação, minimização, maximização, movimento ou redimensionamento de uma janela; antes de completar um comando do sistema; antes de remover um evento de mouse ou teclado da fila de mensagem do sistema; antes de setar o foco da janela; ou antes de sincronizar com a fila de mensagem do sistema[RICH92]. A Tab. B.11 resume os valores esperados por ***LRESULT CALLBACK CBTProc(nCode, wParam, lParam)*** e os valores de retorno esperados pelo *Windows*.

| *Cod* | *wParam* | *lParam* | *lResult* |
|---|---|---|---|
| *HCBT_ACTIVATE* | "*Handle*" da janela que está sendo ativada | aponta para uma estrutura *CBT_ACTIVATE-STRUCT* | Zero (0) se a janela estiver sendo ativada, caso contrário 'um'. |
| *HCBT_CREATEWND* | "*Handle*" da janela que está sendo criada | aponta para uma estrutura *CBT_CREATEWND.* | Zero (0) se a janela estiver sendo criada, caso contrário 'um'. |
| *HCBT_DESTROYWND* | "*Handle*" da janela que está sendo destruída. | NULL | Zero (0) se a janela estiver sendo destruída, caso contrário 'um'. |

***Tab.B.11 Detalha os parâmetros passados pela função filtro do Hook de Hardware***

| Code | wParam | lParam | lResult |
|---|---|---|---|
| *HCBT_MINMAX* | "*Handle*" da janela que está sendo minimizada ou maximizada. | *LOWORD* especifica um valor mostrado (SW_) | Zero (0) se a janela estiver sendo minimizada ou maximizada, caso contrário 'um'. |
| *HCBT_SET-FOCUS* | *Handle* da janela que está ganhando o foco. | *LOWORD* especifica o handle da janela que está perdendo o foco. | Zero se o foco de entrada estiver sendo trocado, caso contrário 'um'. |
| *HCBT_SYSCOMMAND* | Especifica o comando do sistema selecionado pelo usuário. | Se wparam é *SC_HOTKEY* *LOWORD* é o handle da janela que se tornará ativa. Se *wParam* não for *SC_HOTKEY* e o comamdo do sistema foi escolhido com o mouse, *LOWORD* e *HIWORD* representam as coordenadas x e y do cursor do mouse. Para qualquer outra condição, *lParam* é indefinido. | Zero se o comando do sistema estiver sendo executado, caso contrário 'um'. |
| *HCBT_CLI-CKSKIPPED* | Identifica a mensagem do *mouse* removida da fila. | Aponta para a estrutura ***MOUSEHOOKSTRUCT***. | Não utilizado, retorna zero. |
| *HCBT_MO-VESIZE* | "*Handle*" da janela que está sendo movida ou redimensionada. | Aponta para uma estrutura ***RECT***. | Zero se a janela estiver sendo movida ou redimensionada, caso contrário 'um '. |
| *HCBT_QS* | *NULL* | *NULL* | Não utilizado, retorna zero |
| *HCBT_KEYSHIPPED* | Identifica a mensagem do teclado removida | Especifica a mesma informação que é enviada em *lParam* a uma janela quando ela recebe a mensagem ***KEYDOWN***. | Não utilizado, retorna zero |

***Tab.B.11 Detalha os parâmetros passados pela função filtro do Hook de Hardware continuação***

A descrição mais detalhada deste tipo de *hook*, encontra-se no *Help* do *Windows* e em [RICH92] páginas : 391 a 396.

- ***Hook* de mensagem - *WH_SYSMSGFILTER*.**

O *Windows* chama a função filtro deste tipo de *Hook*, se uma mensagem está para ser processada por uma caixa de diálogo, um menu ou *Scrollbar*. O *WH_SYSMSGFILTER* pode ser usado para monitorar mensagens enviadas para qualquer aplicação. Este tipo de *hook* somente pode ter escopo de sistema e, portanto deve está em uma DLL.

A função *LRESULT CALLBACK SysMsgProc(nCode, wParam, lParam)* pode processar ou modificar mensagens para qualquer aplicação dentro do sistema.

**Parâmetros da função filtro do *Hook WH_SYSMSGFILTER*.**

A Tab B.12 descreve os parâmetros passados pela função e os valores de retorno esperado pelo *Windows*.

| *nCode* | *wParam* | *lParam* | lResult |
|---|---|---|---|
| *MSGF_DIALOGBOX* | *NULL* | Aponta para uma estrutura *MSG* | Zero se a mensagem poder ser processada, 'um' se descartada. |
| *MSGF_MENU* | *NULL* | Aponta para uma estrutura *MSG* | Zero se a mensagem poder ser processada, 'um' se descartada. |
| *MSGF_SCROLLBAR* | *NULL* | Aponta para uma estrutura *MSG* | Zero se a mensagem poder ser processada, 'um' se descartada. |
| *MSGF_NEXTMENU* | *NULL* | Aponta para uma estrutura *MSG* | Zero se a mensagem poder ser processada, 'um' se descartada. |

**Tab.B.12 - Detalha os parâmetros da função filtro do *Hook* de Mensagem**

A seguir serão detalhado os parâmetros da função filtro:

nCode - Especifica o tipo de mensagem que está sendo processada. Este parâmetro pode ser:

***MSGF_DIALOGBOX*** Mensagens dentro de uma caixa de diálogo ou a mensagem de um procedimento caixa de diálogo que está sendo processada.

***MSGF_MENU*** Mensagens de teclado ou mouse dentro de um menu que está sendo processado.

Se parâmetro *nCode* é menor do que zero, a função filtro passa a mensagem para a função ***CallNextHookEx*** sem processar seus comandos e retorna o valor retornado pela função ***CallNextHookEx*** para o *Windows*.

*wParam* deve ser *NULL*.

*lParam* Aponta para a estrutura *MSG* para guardar a mensagem. A estrutura *MSG* tem o seguinte formato:

```
typedef struct tagMSG
{
HWND  hwnd;
UINT        message;
WPARAM      wParam;
LPARAM      lParam;
DWORD       time;
POINT  pt;
} MSG;
```

A estrutura *MSG* contém informações da fila de aplicação do *Windows*. A.Tab B.13 , faz uma descrição dos membros desta estrutura:

| Membro | Descrição |
|---|---|
| *hwnd* | Identifica a janela que recebeu a mensagem. |
| *Message* | Especifica o número da mensagem. |
| *Wparam* | Especifica informações adicionais sobre a mensagem. O significado exato depende do valor da mensagem (Membro: *message*). |
| *Lparam* | Especifica informações adicionais sobre a mensagem. O significado exato depende do valor da mensagem (Membro: *message*). |
| *Time* | Especifica a hora na qual a mensagem foi enviada. |
| *Pt* | Especifica a posição do cursor, em coordenadas da tela, quando a mensagem foi enviada. |

**Tab.B.13 Descreve os membros da estrutura *MSG*.**

## B.2- Instalando uma função filtro.

Para acessar um *hook* em particular , uma aplicação instala uma função filtro que processa as mensagens associadas com o hook. A função filtro processa as mensagens antes delas chegarem ao seu destino, que é um procedimento de janela. Uma aplicação instala um *hook* através da chamada a função ***SetWindowsHookEx***. A cada chamada a essa função é adicionada uma nova função filtro ao começo da cadeia. Uma cadeia de funções filtro é uma série de funções filtros conectadas a um tipo particular de *hook* (Fig B.5).

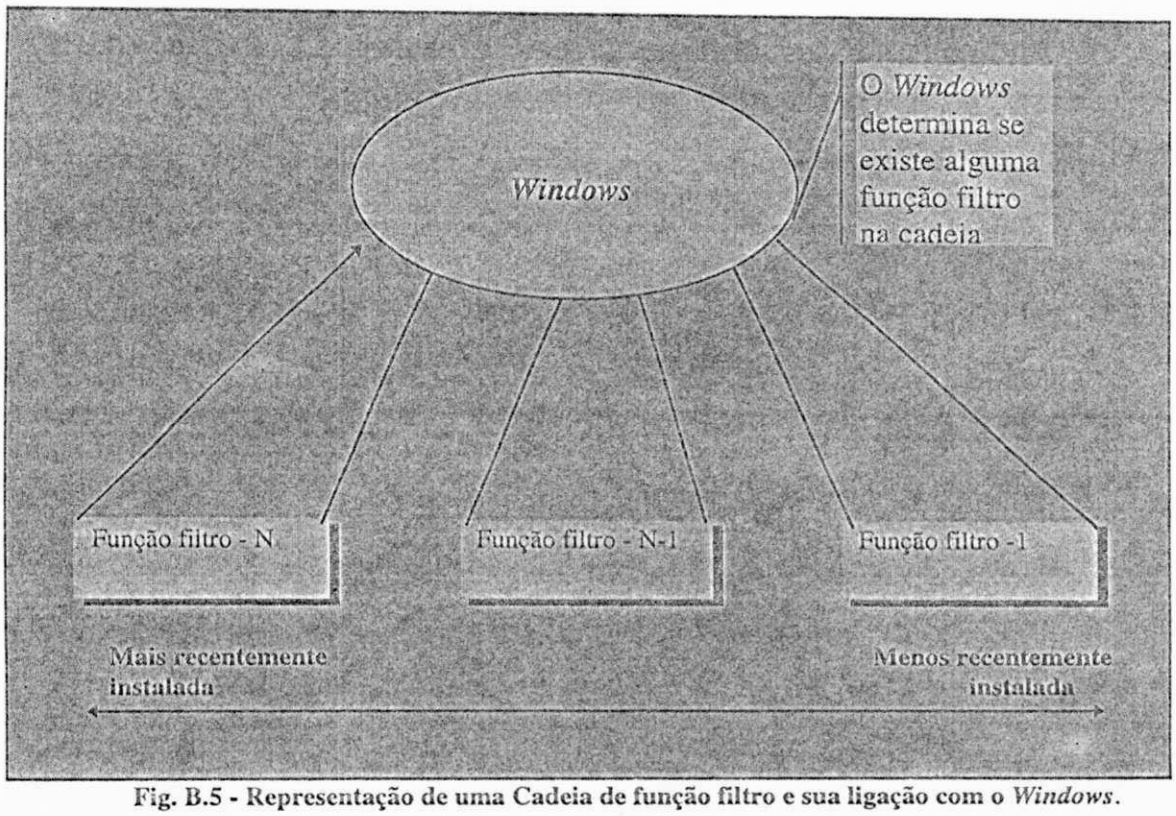



**Fig. B.5 - Representação de uma Cadeia de função filtro e sua ligação com o *Windows*.**

Quando uma aplicação passa o endereço da função filtro para um sistema *hook*, ela deve preservar espaço para o endereço da próxima função filtro na cadeia.

A seguir apresentamos o protótipo da função de instalação de um *hook*.

***HHOOK*** ***SetWindowsHookEx***(*int idHook,HookProc hkPrc,HInstance hInst,HTask hTask*);

Quando esta função é chamada, o *Windows* aloca um bloco de memória que conterá uma estrutura de dados descrevendo um novo *hook* instalado [RICH92]. Cada nova estrutura é colocada em um nodo de uma lista encadeada. Após o nodo ter sido inserido na lista, a função ***SetWindowsHookEx*** retorna um "handle" para o nodo. Este valor de retorno deverá ficar guardado pela aplicação em uma variável global do tipo ***HHOOK***, que é definida no *WINDOWS.H*.

Protótipo da variável:

*static HHOOK _hHook = NULL.*

Onde *_hHook* é o nome dado pela aplicação ao novo tipo de hook instalado. Esta variável será usada quando a aplicação precisar se referir ao hook instalado. Se ocorrer um erro no processo de instalação, esta variável receberá um *NULL*.

Agora, serão descritos os parâmetros que são passados a função ***SetWindowsHookEx.***

☑ *idHook* - Especifica o tipo de *hook* instalado, Tab B.1.

☑ *hkPrc* - Fornece o endereço da instância do procedimento da função filtro. O tipo ***HookProc*** é um apontador de 32 *bits* para um procedimento *hook*. A *hkPrc* segue três regras básicas:

1ª. Se a função filtro está em uma DLL e a função ***SetWindowsHookEx*** está dentro da mesma DLL, o parâmetro é simplesmente o nome da função filtro.

2ª. Se a função ***SetWindowsHookEx*** está em uma aplicação e a função filtro em uma DLL, o parâmetro *hkPrc* pode ser o nome da função filtro. Neste caso a função é prototipada em um arquivo de cabeçalho da DLL e a aplicação deve ligá-la com o arquivo (**.LIB**) associado com a DLL que contém a função filtro. Alternativamente, uma aplicação pode chamar ***GetProcAddress*** para receber o endereço da função filtro que se encontra na DLL e passá-lo para ***SetWindowsHookEx.***

3ª. Se a chamada a ***SetWindowsHookEx*** está em uma Aplicação e a função filtro se encontra na mesma aplicação, o *hkPrc* deve ser o endereço da instância procedural da função filtro. Este endereço é obtido pela chamada a ***MakeProcInstance*** e o parâmetro passado nesta função é o nome da função filtro.

☑ ***hInst*** - é um "*handle*" que identifica a instância do módulo que contêm a função hook. Se a função está em uma aplicação, este valor é o mesmo que foi passado para a função ***WinMain*** da aplicação, dentro do parâmetro *hInstCurrent*. Se a função hook está em uma DLL, este valor é o mesmo que foi passado para a função ***LibMain*** da DLL, dentro do parâmetro *hInst*.

☑ ***hTask*** - é um "*handle*" da instância da tarefa. Este parâmetro diz ao *Windows* se a função *hook* poderá ser chamada de um evento que está ocorrendo em qualquer

parte do sistema ou se apenas de uma tarefa específica. Se uma aplicação quer instalar um hook para uma dada tarefa, ela pode chamar *GetCurrentTask* para obter o "*handle*" da instância da tarefa ou a aplicação pode chamar *GetWindowsTask* para determinar o "*handle*" da instância de uma aplicação que criou uma janela específica. Se uma aplicação quer instalar um *hook* com escopo do sistema, o valor da *hTask* deverá ser *NULL*. Se uma função hook tiver o escopo de sistema, o código da função deverá estar em uma DLL. A função que têm escopo de uma tarefa pode ter seu código dentro de um executável ou em uma DLL.

## B.3- A função filtro.

A função filtro *hkPrc* usada em *SetWindowsHookEx* deve ser declarada, com os mesmos parâmetros ,independente do tipo de *hook* que for instalado [RICH92].

A seguir o protótipo da função filtro:

*LRESULT CALLBACK HookProc hkPrc(int nCode, WPARAM wParam LPARAM lParam);*

Passamos a descrever os parâmetros que são passados pela função.

☑ *nCode* - A estrutura da função filtro do *hook* é semelhante a da função do procedimento da janela no *Windows*.

O valor de *nCode* identifica o tipo de ação (mensagem) que é recebida pela função. Os valores para este parâmetro dependem do tipo de *hook* instalado (Tab B.1.).

☑ *wParam* e *lParam* - seus valores dependem do valor passado por *nCode*

Uma cadeia de funções filtro (FigB.5) é formada quando muitas aplicações instalam hooks de um mesmo tipo. Quando um evento ocorre ligado ao *hook* instalado, o *Windows* chama a função filtro mais recentemente instalada para aquele tipo de *hook*. Ficando a responsabilidade das chamadas às demais funções da cadeia com a aplicação que instalou a última função filtro [RICH92]. Isto é, é responsabilidade de cada função filtro fazer a chamada à função filtro que foi instalada antes dela. Para executar esta

chamada, a função filtro que está ativada, deve, após executar seus comandos, fazer a chamada a função *CallNextHookEx*, cujo protótipo é o seguinte:

*LRESULT CallNextHookEx(HHOOK _hHook,int nCode,WPARAM wParam,LPARAM lParam);*

Esta função chama a próxima função dentro da cadeia, e passa os mesmos parâmetros que recebeu e na mesma ordem, e como primeiro parâmetro o "*handle*" que foi retornado quando da chamada a função *SetWindowsHookEx*. A finalidade da passagem do "*handle*" da função filtro que está sendo executada é permitir que o *Windows* encontre o nodo dentro da cadeia e possa identificar qual a próxima função a ser executada. Se for determinado que não há mais função filtro na cadeia a ser executada a função *CallNextHookEx* retorna um *NULL*.

Se desejarmos que o restante das funções filtros não seja executado, devemos neste caso, colocar nosso próprio valor de retorno. Por outro lado se desejarmos que as demais funções filtro sejam executadas antes das nossa função, devemos colocar a função *CallNextHookEx* no topo da nossa função filtro, e quando terminarmos de executar a nossa função , devemos retornar nosso próprio valor ou o valor retornado pela função *CallNextHookEx*.

O valor de retorno esperado da função filtro depende do tipo de *hook* instalado e do valor do parâmetro *nCode*.

## B.4- Removendo uma função filtro da cadeia

Antes de terminar, uma aplicação deve liberar os recursos do sistema associados com o *hook*, para tanto deve remover a função filtro da cadeia. Isto é feito chamando a função *UnhookwWindowsHookEx* [RICH92].

O protótipo dessa função é:

*Bool UnhookWindowsHookEx(HHOOK _hHook);*

Onde o parâmetro *_hHook* especifica o "*handle*" do *hook* que se deseja remover. Ovalor deste parâmetro é o valor que foi retornado quando da instalação do *hook* pela

função ***SetWindowsHookEx***. Se a função de remoção do hook retornar um valor diferente de zero (0) podemos afirmar que foi obtido sucesso na remoção.

O *Windows* não requer que a função filtro seja desativada na mesma ordem em que foi instalada. No entanto quando uma função é retirada da cadeia , o *Windows* destroi o bloco de memória que identificava o nodo, referente a função filtro que está sendo removida, e atualiza os ponteiros da lista encadeada (Fig B.5).