

UNIVERSIDADE FEDERAL DE UBERLÂNDIA  
FACULDADE DE ENGENHARIA ELÉTRICA  
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UMA INTERFACE GRÁFICA COMPACTA E  
PRÁTICA EM *OBJECTIVE CAML*

João Barbosa Souza Filho

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciências.

Área de concentração: Processamento da Informação.

Linha de pesquisa: Inteligência Artificial.

Orientador: Prof. Dr. Antonio Eduardo Costa Pereira

Uberlândia

Julho de 2007

João Barbosa Souza Filho

Uma Interface Gráfica Compacta e Prática em  
*Objective Caml*

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciências.

Área de concentração: Processamento da Informação.

Linha de pesquisa: Inteligência Artificial.

Uberlândia, 25 de Julho de 2007

Composição da Banca Examinadora:

|            |                               |                |   |     |
|------------|-------------------------------|----------------|---|-----|
| Prof. Dr.  | Antonio Eduardo Costa Pereira | Orientador     | - | UFU |
| Prof. Dr.  | Alexsandro Santos Soares      | Membro Externo | - | UFG |
| Prof. Dra. | Elise Barbosa Mendes          | Membro         | - | UFU |

# Agradecimentos

- Agradeço ao meu orientador Prof. Costa pelo apoio e conhecimento adquirido durante a trajetória deste trabalho que irá me acompanhar pela vida.
- Aos professores Luciano, Alexsandro, Adriano Andrade, Alcimar, Edgar, Alexandre e Edna por me ensinaram conceitos importantes que contribuíram para este trabalho;
- Ao professor Keiji que me inspirou em vários momentos, além de seu ensino e convívio edificante;
- Aos colegas e amigos Junia e Reny que acompanharam e colaboram neste trabalho;
- Aos amigos que direta e indiretamente contribuíram para este trabalho.
- À minha família por sua compreensão pela minha ausência de seu convívio.
- Em especial aos meus filhos por suportarem minha ausência.
- À Kheline pelo amor, apoio e incentivo, indispensáveis à minha vida.

*"Play: Work that you enjoy doing for nothing. "*

— EVAN ESAR

# Resumo

O *OCaml* e as outras linguagens da família do *ML* não fornecem ao programador nenhuma interface gráfica nativa. Por isso, torna-se necessário recorrer a interfaces gráficas terceirizadas como *TCL/TK* e *GTK*. Em consequência disto, o usuário de programas escritos na referida linguagem precisam pesquisar na internet para encontrar a biblioteca necessária à execução dos programas que deseja utilizar, efetuar o *download* e instalá-la. A maioria dos usuários finais não possuem conhecimento necessário ou paciência para efetuar esta tarefa, razão pela qual o objetivo deste trabalho foi a criação de uma interface gráfica inteiramente escrita em *OCaml*, que não exige biblioteca externa para funcionar, além de poucos recursos da máquina utilizada. Programas compilados com esta ferramenta ocupam pouco mais de 300 KBytes no disco. A interface opera igualmente bem no Windows, Linux ou demais sistemas operacionais onde *OCaml* foi portado. Ela apresenta quatro funcionalidades, a saber, botões, que podem ser gráficos ou textuais, dispositivos de restauração da tela, processadores de linguagem e editores com funcionamento similar ao do *Emacs*, os quais podem trabalhar com máscaras ou com textos livres, já que *OCaml* é uma linguagem fortemente tipada e funcional, cuja interface gráfica descrita é praticamente livre de *bugs*.

# Abstract

OCaml and other ML like languages does not provide developer with a native GUI. Therefore one must use a third party GUI, like TCL/TK or GTK. This requires final users to search the Internet for a GUI library, download it, install it, etc. Most final users are not up to the task. Therefore we decided to create a GUI interface entirely written in OCaml, that does not require any external library to work, and that has a very small footprint. Compiled programs occupy as little as 300Kbytes on disk. The system works equally well on Windows, Linux, or any other operational system with an OCaml port. It has four kinds of widgets: Buttons, refresh devices, parsers, and Emacs-like mini-editors. The Emacs widgets work with plain text or mask-driven forms; they offer a practical and simple method for data input. Since OCaml is a strong typed functional language, this application is practically bug free.

# Sumário

|                                                 |    |
|-------------------------------------------------|----|
| LISTA DE FIGURAS . . . . .                      | ix |
| 1 INTRODUÇÃO . . . . .                          | 10 |
| 1.1 Histórico . . . . .                         | 13 |
| 1.2 Dificuldades para o Usuário Final . . . . . | 16 |
| 1.3 Dificuldades para o Desenvolvedor . . . . . | 18 |
| 1.4 Instalação de Bibliotecas . . . . .         | 20 |
| 1.5 Objetivos deste Trabalho . . . . .          | 23 |
| 1.6 Motivação . . . . .                         | 23 |
| 2 METÁFORAS . . . . .                           | 25 |
| 2.1 Metáfora da Mesa de Trabalho . . . . .      | 28 |
| 2.2 Metáfora das Pastas e Documentos . . . . .  | 28 |
| 2.3 Metáfora dos Formulários . . . . .          | 30 |
| 2.4 Metáfora de Console . . . . .               | 30 |
| 2.5 Metáfora do Menu . . . . .                  | 32 |
| 2.6 Metáfora do Painel de Controle . . . . .    | 34 |
| 2.7 Metáfora do Ícone . . . . .                 | 35 |
| 3 INTERFACE GRÁFICA DE USUÁRIO . . . . .        | 37 |
| 3.1 O que é Usabilidade? . . . . .              | 39 |

---

|     |                                             |    |
|-----|---------------------------------------------|----|
| 4   | BIBLIOTECA PROPOSTA . . . . .               | 43 |
| 4.1 | Criando botões . . . . .                    | 46 |
| 4.2 | Criando um Editor . . . . .                 | 49 |
| 4.3 | Ativando os Objetos . . . . .               | 50 |
| 4.4 | Criando Botões Gráficos . . . . .           | 50 |
| 4.5 | Criando Formulários . . . . .               | 51 |
| 4.6 | Publicação e Distribuição . . . . .         | 52 |
| 4.7 | Considerações Finais . . . . .              | 52 |
| 5   | CONSIDERAÇÕES FINAIS E CONCLUSÕES . . . . . | 53 |
| 5.1 | Futuras Contribuições . . . . .             | 55 |
|     | REFERÊNCIAS BIBLIOGRÁFICAS . . . . .        | 58 |

# Lista de Figuras

|                                                                                                                           |    |
|---------------------------------------------------------------------------------------------------------------------------|----|
| FIGURA 1.1 – <i>Desktop</i> Mac OS . . . . .                                                                              | 12 |
| FIGURA 1.2 – <i>Desktop</i> do Linux . . . . .                                                                            | 12 |
| FIGURA 1.3 – <i>Desktop</i> do Windows . . . . .                                                                          | 13 |
| FIGURA 1.4 – Estação de trabalho Star da Xerox, a primeira implementação co-<br>mercial de uma interface gráfica. . . . . | 14 |
| FIGURA 1.5 – Estação de trabalho Lisa, a primeira implementação de interface<br>gráfica da Apple. . . . .                 | 14 |
| FIGURA 1.6 – Estação de trabalho Macintosh, da Apple, o primeiro sucesso comer-<br>cial com interface gráfica. . . . .    | 14 |
| FIGURA 1.7 – Tela de Console com um Menu do programa Fdisk do Windows 98 .                                                | 15 |
| FIGURA 1.8 – Imagem da primeira versão (original) do <i>Desktop</i> do Macintosh . . .                                    | 16 |
| FIGURA 1.9 – Arrastar um arquivo para a lixeira . . . . .                                                                 | 18 |
| FIGURA 2.1 – Imagem do <i>Desktop</i> do Windows versão 3.11 em 1985 - Metáforas<br>da Mesa de Trabalho . . . . .         | 29 |
| FIGURA 2.2 – Documentos e Pastas no Windows XP - Metáforas das Pastas e<br>Documentos . . . . .                           | 30 |
| FIGURA 2.3 – Tela de console do MSDOS v 6.0 - Metáfora do Console . . . . .                                               | 31 |
| FIGURA 2.4 – Menu de Tela Inteira - Metáfora do Menu . . . . .                                                            | 32 |
| FIGURA 2.5 – Menu suspenso - Metáfora do Menu . . . . .                                                                   | 33 |
| FIGURA 2.6 – Exemplo de Botões . . . . .                                                                                  | 34 |

---

|                                                                                                                                                                                                  |    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| FIGURA 2.7 – Exemplo de Botões com Imagens . . . . .                                                                                                                                             | 34 |
| FIGURA 2.8 – Exemplo de botão de checagem ( <i>check box</i> ) . . . . .                                                                                                                         | 35 |
| FIGURA 2.9 – Exemplo de botão caixa de rádio ( <i>radio box</i> ) . . . . .                                                                                                                      | 35 |
| FIGURA 2.10 – Ícones - Metáfora do ícone . . . . .                                                                                                                                               | 36 |
| FIGURA 3.1 – Processo de Desenvolvimento de Interface . . . . .                                                                                                                                  | 39 |
| FIGURA 4.1 – Programa exemplo gerado pela Listagem 1 . . . . .                                                                                                                                   | 48 |
| FIGURA 4.2 – Imagem do formulário gerado a partir do programa nw.ml. . . . .                                                                                                                     | 51 |
| FIGURA 4.3 – Imagem da localização geográfica dos usuários que acessaram o <i>site</i><br>da biblioteca LEMAC através do <i>site</i> Sourceforge.net, obtido em 23<br>de agosto de 2007. . . . . | 52 |

# 1 Introdução

As ações pelo desenvolvimento da tecnologia computacional ocorreram de forma natural, em que os sistemas de uso pessoal deixaram as primeiras marcas de utilização, desenvolvendo-se para a assimilação da interface gráfica. Tão natural, desejada e necessária quanto se poderia imaginar. Assim afirma Dondis, "a compreensão visual é um meio natural que não precisa ser aprendido."(DONDIS, 1991)

O computador só precisou de avanços tecnológicos suficientes (maior resolução de tela, maior capacidade de processamento e de memória) para passar a apresentar gráficos e imagens. Os sistemas computacionais, que envolvem a utilização das imagens como recursos para a navegabilidade deles têm obtido maior sucesso nos dias atuais. Ainda que a utilização de textos tenha sido firmada, os recursos que possam mesclar ambas (textos e imagens) com certeza têm se difundido mais e mais, como vemos em Dondis, "para os analfabetos, a linguagem falada, a imagem e o símbolo continuam sendo os principais meios de comunicação e, dentre eles, só o visual pode ser mantido em qualquer circunstância prática. Isso é tão verdadeiro hoje quanto tem sido ao longo da história."(DONDIS, 1991). No universo computacional, a grande massa de usuários é também analfabeta de algum modo: em relação ao funcionamento da máquina e de suas linguagens de mais baixo nível, o que é destacado por Júlio Plaza: "A imagem e a linguagem visual estão sendo atualmente privilegiadas pela informática, em parte devido ao caráter de condensação e síntese, que permite compreendê-la rapidamente fazendo jus à máxima: 'Uma imagem vale por mil palavras'. Toda sorte de imagens, diagramas, metáforas e gráficos tratados pela informática, ajudam o pensamento visual e verbal."(PLAZA, 1991)

Como os acontecimentos no mundo da tecnologia são rápidos, a utilização do computador como ferramenta de uso diário das pessoas, passa a ter maior impacto, apontando para esta tecnologia um novo conceito. Nesse parâmetro entende-se que a realidade foi modi-

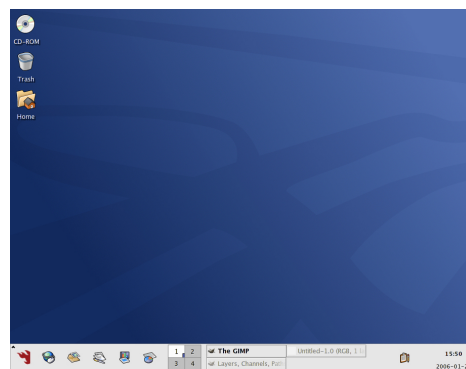
ficada, e a necessidade de uma proposta pela facilidade dos acessos entre a máquina e o usuário, também teve que ser revista. Surge a interatividade como recurso imprescindível para tal.

Os padrões puderam ser revistos uma vez que os monitores e as placas controladoras de vídeo já dispunham de tecnologias suficientes para tal proposta, como cores, processadores de imagens, gráficos vetoriais, resolução cada vez maior e outros, o que resultou em condições suficientes para tornar injustificável a necessidade de usuários terem que decorar uma extensa lista de comandos e a possibilidade de um ambiente gráfico. Percebendo essa nova realidade, as empresas Apple e Xerox foram pioneiras em criar e desenvolver o que se convencionou chamar "interface amigável ao usuário" (*user friendly interface*) ou "interface gráfica de usuário" (*graphic user interface* - GUI), duas maneiras de se denominar o ambiente gráfico amigável apresentado pelos sistemas operacionais atuais, que usamos diariamente como exemplos o Windows, o Mac OS e o X Window no Linux.

Nas interfaces gráficas o uso de componentes visuais (*widgets*) que imitam objetos físicos, tais como: botões; janelas; quadros; formulários; chaves e campos de anotação, entre outros, pode acelerar o aprendizado mediante programas que se associam com seu paralelo do mundo real, em que as mesmas ações e reações são esperadas em seu clone computacional. Por serem interativas, as interfaces gráficas adquirem uma dimensão a mais. Ted Nelson, um dos estudiosos de sistemas interativos, "percebeu a sensação de satisfação nos usuários, quando uma interface é construída pelo que ele chamou de princípio da virtualidade - uma representação da realidade que pode ser manipulada."(SCHNEIDERMAN, 1983).

Existem problemas a serem resolvidos e ações a serem melhor definidas, quando se trata da utilização das interfaces gráficas, pois estas nem sempre são simples ou de fácil assimilação. Há na atualidade grande variedade de programas comerciais com interface de difícil aprendizado, ou com apresentação de resultados com problemas quanto à interpretação, pela utilização inadequada de componentes visuais ou pelo excesso de seu uso.

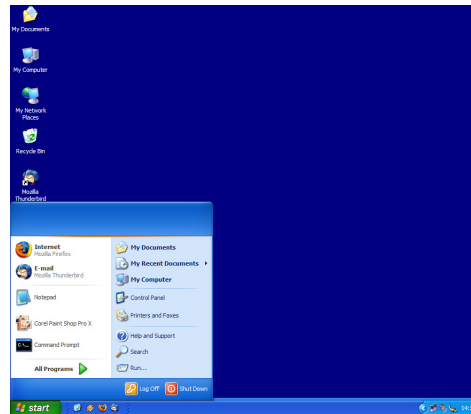
Tais questões requerem do usuário tempo excessivo para o aprendizado, com altos índices de equívocos na operação, ou passos desnecessários na sua execução, o que pode tornar a interface ineficiente ou pouco produtiva.

FIGURA 1.1 – *Desktop* Mac OSFIGURA 1.2 – *Desktop* do Linux

Um bom projeto de interface gráfica requer que o usuário não precise memorizar muitos passos ou controles para realizar uma ação. Isto é particularmente importante para programas científicos, que têm como objetivo compreender os conceitos que o programa tem em sua base e não aprimorar ou dominar o conhecimento de como realizar os passos para executar tal ou qual operação.

No entanto, usuários experientes podem desejar a utilização de teclas de atalho (*shortcuts*) ou a digitação de comandos para a realização de algumas tarefas, utilizando a memória e assim conseguir mais produtividade e agilidade, mesmo este recurso não sendo intuitivo (BARRIER, 2002)(CHELARU, 2007).

A proposta necessária é ter um equilíbrio das funcionalidades e oferecer poucos recursos visuais, que estejam alinhados com o que os usuários esperam da interface e, ao mesmo tempo, fornecer controles aceleradores de utilização dos programas como as teclas de atalho. Ponderar e testar a interface é o ideal.

FIGURA 1.3 – *Desktop* do Windows

## 1.1 Histórico

Antes das interfaces gráficas, os usuários finais utilizavam uma tela de texto, na qual digitavam-se comandos e recebia-se sua resposta (tela de console). Para controlar o computador por meio do teclado, ainda há nos dias de hoje programas que mantêm menu em modo texto, como o Fdisk, do sistema operacional Linux, e mesmo o Windows (FREITAS, 2005).

Ao se mencionar sobre interfaces gráficas lembra-se de Vannevar Bush, que na década de 30 já discorria suas idéias sobre elas com um dispositivo chamado "Memex", que funcionava como uma espécie máquina com duas telas sensíveis ao toque, um teclado e um *scanner*. Com isto seria possível o acesso a todo o conhecimento humano graças a conexões muito semelhantes aos *links* (*hyperlinks*).

Como o computador digital ainda não havia sido desenvolvido, suas idéias foram apenas lidas e discutidas (BUSH, 1945). A primeira implementação de uma interface gráfica foi realizada por Ivan Sutherland, do MIT (Massachusetts Institute of Technology) que em 1962 desenvolveu o *SketchPad*, um programa que permitia aos usuários criarem linhas, círculos e pontos (SUTHERLAND, 1963). No entanto, o primeiro sucesso comercial só veio muitos anos depois com o Apple Macintosh, em 1984, baseado no seu precursor de 1983, o Apple Lisa, que, por outro lado, foi criado a partir do sistema da Xerox, o Star, do início dos anos 70 (BENGOCHEA, 2005), (BREY, 2005) e (BRAGA, 2004).

Desde que o computador pessoal Macintosh chegou ao mercado em 1984, os usuários passaram a aspirar por interfaces gráficas, as quais a princípio eram mais simples de se



FIGURA 1.4 – Estação de trabalho Star da Xerox, a primeira implementação comercial de uma interface gráfica.



FIGURA 1.5 – Estação de trabalho Lisa, a primeira implementação de interface gráfica da Apple.



FIGURA 1.6 – Estação de trabalho Macintosh, da Apple, o primeiro sucesso comercial com interface gráfica.

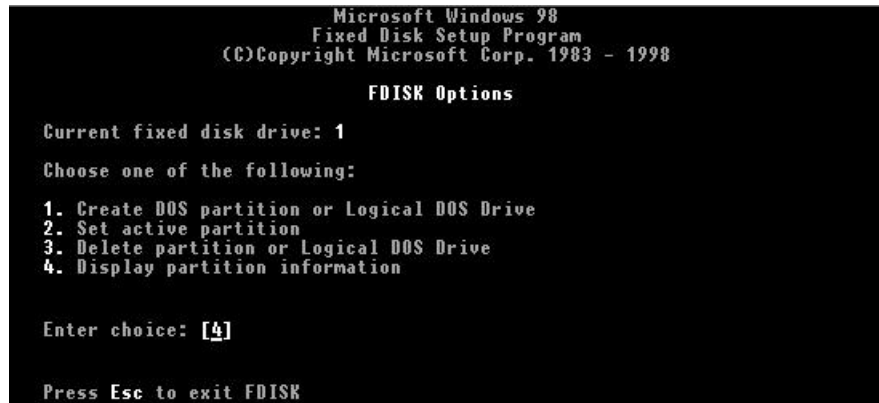


FIGURA 1.7 – Tela de Console com um Menu do programa Fdisk do Windows 98

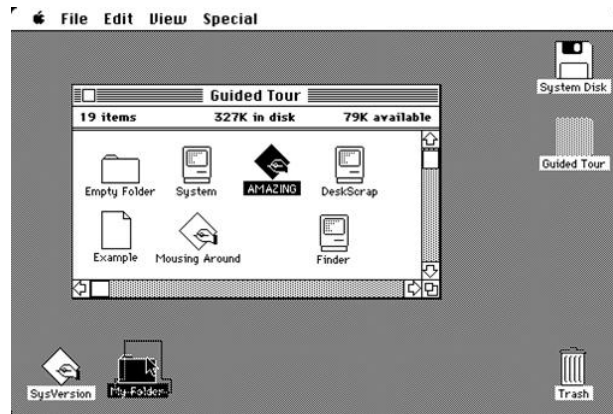
utilizar em relação aos outros sistemas existentes na época, baseados em telas de texto, chamadas de *console* (Figura 1.7).

Mas o sucesso do produto da Apple não está em apenas criar um ambiente gráfico com janelas e botões e, sim, na metodologia de desenvolvimento do sistema como um todo. O Macintosh foi construído a partir de um modelo conceitual que leva em conta o relacionamento entre o usuário e o sistema. Este é representado por um ambiente de trabalho em um escritório, mais tarde chamado de "metáfora do tampo de escrivaninha" (*desktop metaphor*).

Na tela são mostrados ícones de objetos de uso comum em uma empresa, como documentos, pastas, arquivos, lixeiras, etc. No entanto, a natureza visual de uma interface gráfica não é o seu maior trunfo. Ela traz à mente do usuário todo um conjunto de relações que existe entre os objetos que compõem a interface, toda uma série de procedimentos e ações que fazem parte do modelo representado e que, portanto, são familiares.

Com o tempo as empresas de *software* começaram a sofrer pressões dos usuários finais para adotarem os últimos avanços conquistados por seus concorrentes, com propostas de novos recursos que, na opinião do proponente, tornam a interação com o usuário mais amigável, prática ou familiar.

Com isso a tela do computador passou a ser povoada por janelas cheias de dispositivos de controle denominados componentes visuais (*widgets*) dentre os quais figuram botões, menus que se desenrolam, árvores de opções, campos de edição, pastas e vários outros. Todos estes componentes visuais deveriam ser ativados pelo *mouse* ou outro dispositivo

FIGURA 1.8 – Imagem da primeira versão (original) do *Desktop* do Macintosh

apontador. Acontece que muitos usuários experientes sentiam necessidade dos menus controlados por teclado, razão pela qual surgiram as teclas de atalho. Daí eles utilizavam as telas de console, por entenderem que a possibilidade de digitar comandos tornava tal acesso mais rápido e mais direto, uma vez que procurar em janelas e pastas a aplicação ou o componente visual que se desejavam disparar, em certos momentos, torna o processo mais complexo e demorado. Assim as interfaces gráficas receberam não apenas consoles, mas também *macros* e linguagens de *script*, em tudo semelhantes às antigas linguagens de processamento *batch*.

Interessante notar que os programas mais populares da atualidade utilizam as interfaces gráficas apenas para chamar uma console. Mesmo assim, grande parte dos usuários considera as interfaces gráficas mais amigáveis. Um exemplo concreto esclarece este ponto: o usuário utiliza um menu ou um ícone na barra de ferramentas para disparar um programa de bate-papo (exemplo do MSN da Microsoft), o qual na verdade é uma console que em nada difere das em uso no início da década de 80.

## 1.2 Dificuldades para o Usuário Final

Os componentes visuais foram projetados para fazer uma correspondência entre o que o usuário vê na interface e o que ele deve pensar sobre o significado do que ele vê. Em vez de pensar no próprio sistema representado pela interface, este transporta o usuário do sistema para um domínio familiar. O efeito disto é que o usuário irá desenvolver um modelo mental do sistema que estará muito mais próximo do mundo assemelhado do que

do mundo no qual o sistema representa. (CARROL; MARK; KELLOG, 1988) afirmam que um problema de *design* surge quando a metáfora não capta certas funcionalidades exigidas pelo sistema a ser representado na interface. Neste caso, os *designers* devem combinar a metáfora original com outras metáforas, criando o que (CARROL; MARK; KELLOG, 1988) chamam de metáforas compostas. Mas as metáforas compostas introduzem um novo problema. (TOGNAZZINI, 1992) descreve que as pessoas têm dificuldade em saber o que elas devem fazer quando estão lidando com várias metáforas diferentes, em uma mesma interface.

Por outro lado, (ECO, 1971) considera que o aspecto negativo é a própria abundância de informações e recursos: "O excesso pode ser péssimo, porque não se consegue encará-lo e escolher o que presta." Navega-se, portanto por um dilúvio de dados e controles. Temos a sensação de que informações excessivas são o mesmo que não tê-las. Já que "navegar, em linhas gerais é a arte de encontrar um caminho que leve de um lugar a outro". (LEÃO, 1999)

Em resumo, as interfaces gráficas modernas tornaram-se de difícil utilização porque os fabricantes começaram a dotá-las de uma profusão de controles visuais e recursos redundantes (COMPUTER, 1989), (CABRAL, 2007). Por exemplo, para disparar um editor de textos, pode-se usar:

- Um ícone, que pode estar no *desktop* de uma barra de ferramentas, de uma árvore de navegação, de uma janela de pasta, de um navegador, da barra de *status*, dentre outros;
- Um menu, que pode ser o menu "Iniciar", um menu destacável (*popup*) disparado com o botão direito do *mouse*, o de janela de pasta, o do navegador e outros;
- Um console, que pode ser do sistema ou disparado de um aplicativo de busca;
- Um diálogo, como é o caso da caixa de diálogo do comando "executar", no menu "Iniciar".

Contudo, não é fácil para o usuário final aprender ou reconhecer todas essas opções, nem é fácil encontrar a sequência correta de ativações dos recursos visuais tais como: menus, ícones e caixas de diálogo, entre tantas de opções.



FIGURA 1.9 – Arrastar um arquivo para a lixeira

Assim, o comportamento inconsistente dos diversos dispositivos de controle torna difícil a aprendizagem e leva a freqüentes desastres cibernéticos, com perda de informação a exemplo das formas abaixo de compactar uma pasta:

- Chamar o programa compactador da linha de comando, com argumentos dos nomes da pasta a ser compactada e do arquivo resultante da compactação.

Ex.: PKZIP documentos doc.zip;

- Abrir um programa compactador e apertar a seqüência correta de botões, ícones, árvores de navegação e assim por diante;
- Arrastar a pasta até o ícone do compactador.

Com freqüência, o usuário arrasta a pasta não para o ícone do compactador, mas para outros programa a exemplo da lixeira. Neste caso, a pasta torna-se inacessível. No entanto, basta que a arraste para outra pasta, na qual não tenha direitos de acesso, para que a pasta arrastada fique inacessível. O problema consiste nas diferentes semânticas da ação de arrastar a pasta, pois de fato pode-se arrastá-la para transferí-la, compactá-la, gravá-la e criptografá-la, por exemplo.

### 1.3 Dificuldades para o Desenvolvedor

Na programação visual, utilizam-se ferramentas como ícones, containeres e diagramas que representam os vários componentes da interface, as quais permitem que o projetista

construa uma representação visual de como será a interface. A grande falha da programação visual é que ela é de difícil documentação, ou seja, impede a descrição precisa da estrutura e do funcionamento da interface.

A programação visual também é dependente da sequência do desenvolvimento da referida aplicação, cujo resultado depende dos passos e da ordem da montagem efetuada pelo desenvolvedor. Se um módulo ou um componente for acrescentado na ordem errada, nem a interface nem o programa funcionarão, pois não existem ferramentas de fácil uso para documentar a construção de interfaces visuais: em geral, elas são de difícil aprendizado, utilização e, principalmente, documentação.

Com tais pareceres entende-se que novos desafios surgem para o programador como portar seus programas de modo que possam ser compilados em várias plataformas. Ou seja, que o código seja multiplataforma, porém não é a única. Outras características que freqüentemente atrapalham os desenvolvedores incluem (mas podem não estar limitadas a) (MCKAY, 1997):

- Alinhamento de *byte* (*little-endian* x *bigendian*) de tipos de dados em arquivos e conexões de redes;
- Formatos de dados (Ex.: ponto flutuante que não seja IEEE);
- Gerenciamento de memória (Ex.: limitações de segmentos de 64k);
- Fim de linha em arquivos de texto;
- Navegação entre diretórios, pastas e serviços de gerenciamento de arquivos;
- Suporte a *multi-threading*;
- Comunicações entre processos;
- Funções específicas de um sistema específico (Sistema operacional ou biblioteca).

Quando se desenvolve uma aplicação que possa ser portada para mais de uma plataforma (mesmo se as plataformas forem as versões de 16, 32 ou 64 *bits* do Windows), deve-se ser cauteloso com as diferenças entre as plataformas antes que o desenvolvimento comece.

De fato, mesmo programadores experientes têm dificuldades em criar interfaces gráficas. A abertura de uma janela exige dezenas de parâmetros que controlam processos, criação de identificadores de dispositivos, descrição de componentes, funções restauradoras de aspectos, posicionamento de componentes e grades, dentre muitos outros.

Para facilitar a programação, duas classes de ferramentas foram criadas: a primeira consiste na programação visual de interfaces e a outra, objetos de Entrada/Saída, objetivo deste trabalho.

Os objetos de Entrada/Saída utilizados em linguagens como Java e Clean permitem uma boa documentação e manutenção do sistema. Entretanto, as ferramentas baseadas nessa modalidade exigem uma programação extensa e enorme esforço de memória, pois o programador necessita descrever em texto não apenas a estrutura da interface, mas cada aspecto inerente ao seu comportamento dinâmico.

Já na programação visual uma interface gráfica é criada arrastando-se ícones de objetos para uma janela e posicionando-os para se ter noção do visual final da aplicação. Esta aparente facilidade na criação da interface gera dificuldades de documentação do sistema e um correto controle das possíveis interações com o usuário.

## 1.4 Instalação de Bibliotecas

Outro problema peculiar às interfaces gráficas tradicionais é que em várias delas existe a necessidade da instalação de uma grande diversidade de componentes de programas e bibliotecas de rotinas específicas. Este problema não se limita à portabilidade dos aplicativos entre fabricantes de diferentes interfaces ou sistemas operacionais, pois mesmo se restrito a um único fabricante, como exemplo a distribuição de Linux SUSE, existem pelo menos quatro sistemas de interfaces diferentes e no mínimo, dez sistemas de bibliotecas de interfaces gráficas diferentes.

Citam-se a seguir exemplos de algumas bibliotecas gráficas disponíveis na *Internet* e nos sistemas operacionais comerciais, obtidos em pesquisa pela *Internet*, em agosto de 2007.

- Bibliotecas de interfaces gráficas integradas ao sistema operacional:

| Sistema Operacional | Biblioteca     |
|---------------------|----------------|
| Windows             | Windows API    |
| Mac OS X            | Carbon         |
| Mac OS (clássico)   | Mac OS toolbox |

Convém lembrar que nesse caso não há a necessidade de instalar bibliotecas de rotinas de manipulação de interface gráfica, pois estas rotinas já serão instaladas junto com a instalação do próprio sistema operacional. Sua desvantagem está no fato de que essas rotinas funcionam apenas para o ambiente operacional para as quais foram desenvolvidas e não permitem que programas que as utilizem tenham portabilidade. Assim essas aplicações ficarão restritas a esses ambientes, o que não é vantajoso. Para o desenvolvedor, que ficará restrito a apenas uma plataforma e também para o usuário, que só poderá utilizar o programa se tiver o mesmo sistema operacional no qual o sistema foi projetado.

Alguns exemplos de bibliotecas de interfaces gráficas multiplataforma:

- GTK+ (*runtime*)
  - Ambientes: Windows, Linux, X11, Mac OS
  - Endereço de *Internet*: <http://www.gtk.org/download/>
  - Tamanho de *download* (versão 2.10.13): 6.4 Mb
  - Tamanho instalado (versão 2.13): 29.2 Mb
- wxWidgets
  - Ambientes: Windows, OS/2, OpenVMS, Mac OS, Linux, X11
  - Endereço de *Internet*: <http://www.wxwidgets.org/downloads/>
  - Tamanho de *download* (versão 2.8.4): 12,7 Mb
  - Tamanho instalado (versão 2.8.4): 149 Mb
- QT
  - Ambientes: Windows, Linux, X11, Mac OS, PocketPC, WindowsCE
  - Endereço de *Internet*: <http://trolltech.com/developer/downloads/qt/windows>
  - Tamanho de *download* (versão 4.3.1): 69.1 Mb

- Tamanho instalado (versão 4.3.1): 221 Mb
- FOX Toolkit
  - Ambientes: Windows, Linux, X11, Mac OS
  - Endereço de *Internet*: <http://www.fox-toolkit.org/download.html>
  - Tamanho de *download* (versão 1.7.11): 5.2 Mb
  - Tamanho instalado (versão 1.7.11): 14.6 Mb

Essas bibliotecas são multiplataformas, mas não estão incluídas em nenhum dos pacotes de instalação dos sistemas operacionais comerciais da atualidade. Então elas devem ser encontradas na *Internet* ou distribuídas juntamente com o programa que foi desenvolvido para utilizá-la e ser instalada apropriadamente no computador para poder ser utilizada. Esta tarefa não é simples, e até programadores experientes têm dificuldades em realizá-la. Contudo por ser o programador um profissional habilitado, pode-se admitir como sua obrigação saber como resolver tal categoria de problemas.

Uma das questões que o desenvolvedor e o usuário enfrentam é encontrar uma fonte segura e correta para instalar a biblioteca em seu computador, visto que nem sempre seu fabricante possui um *site* oficial na *Internet*. Uma boa busca nos *sites* de pesquisa é necessária. Outra questão é o tamanho dos programas, que consomem cerca de 5 a 70 Mbytes somente nesta etapa. Depois de instalados eles podem chegar a consumir mais de 200 Mbytes de espaço em disco, o que inviabiliza sua utilização em computadores mais simples e com poucos recursos, como os encontrados em regiões pobres e nas escolas públicas do Brasil. O próprio processo de baixar o programa pela *Internet* já significa um esforço a ser executado pelo usuário, visto que nem sempre ele possui uma boa e rápida conexão. Esse processo pode levar horas.

A instalação dessas bibliotecas pode requerer do usuário algumas tarefas extras, como configurar o caminho padrão de pesquisa do sistema operacional (*path*) para que ele as encontre. Isto ocorre com frequência em programas que não possuem um instalador próprio, como no caso do Fox Toolkit, citado neste trabalho. Problemas nas versões das bibliotecas também podem ocorrer, ou seja, se for instalada a versão 1.1 da biblioteca tal e o sistema foi desenvolvido utilizando a versão 2.0 da mesma, normalmente ocorrerão problemas e o programa poderá não funcionar. Também há casos em que mesmo se o

usuário venha a instalar outra versão mais recente ou superior, diga-se a versão 3.4 da mesma biblioteca, o sistema poderá não funcionar: o sistema está dependente da biblioteca e de uma versão específica dela.

## 1.5 Objetivos deste Trabalho

Este trabalho tem como objetivo a construção de uma interface gráfica para a linguagem *Objective Caml* que resolva algumas das questões expostas neste capítulo.

Como objetivos específicos tem-se:

- Ser compacta, ou seja, não deve acrescentar significativamente o tamanho da aplicação;
- Ser portátil de um sistema operacional para outro e não exigir instalação de bibliotecas além das básicas que já estão instaladas no sistema;
- Ser fácil de programar e ampliadora da produtividade do desenvolvedor;
- Ser fácil de aprender a programar para quem não é profissional de informática, para que seja utilizada em educação com auxílio de computador e no desenvolvimento mental de estudantes de diversas áreas;
- Ser fácil de ser utilizada e aprendida por quem apenas deseja utilizar o computador na execução de aplicações finais;
- Demonstrar a utilidade prática das linguagens funcionais na construção de interfaces gráficas.

## 1.6 Motivação

A construção de interfaces gráficas é uma atividade que depende de tempo e esforço do desenvolvedor que muitas vezes dedicará mais tempo em aprimorar os detalhes da interação com o usuário, do que as técnicas por trás da aplicação.

Já o usuário fica atordoado com a utilização de muitos controles diferentes, o que dificulta a eficácia proposta na aplicação em uso. Isto decorre da questão relacionada à

necessidade de maiores conhecimentos ou limitações para operacionalizar computadores, por parte da maioria dos usuários domésticos.

Tal problemática é que define as estruturas de pesquisa deste projeto, e gera para o contexto de pesquisa, desenvolvimento e argumentação dos paradoxos, com o surgimento da nova proposta, a motivação necessária para enfrentar tais desafios. Assim surgem os pareceres descritos nesse contexto, que fundamentados resultam na busca de uma ferramenta que possa simplificar a construção de interfaces gráficas e que estas possam ser produtivas e satisfatórias ao usuário, fornecendo um pequeno conjunto de recursos, suficientemente poderoso para interagir com o usuário.

Desafios técnicos como a portabilidade são outro elemento de grande interesse que ajudam a fugir dos debates sobre qual ambiente ou sistema irá prevalecer, qual moda irá reger o momento (APIKI, 1994) e demonstrar que desenvolvedores podem gerar aplicações acima deste debate e manter seus esforços na sua aplicação.

Apesar da cada vez maior abundância dos recursos computacionais, a utilização razoável destes recursos é uma preocupação que deveria ser permanente na mente de todos os desenvolvedores e que neste trabalho sempre foi muito considerado, permitindo demonstrar que ainda é possível construir aplicações que não desperdicem os recursos computacionais e sejam produtivos.

## 2 Metáforas

O termo *metáfora* é tradicionalmente associado à linguagem humana, mas também é comumente empregado na computação, podendo defini-la, de acordo com o Dicionário Houaiss da Língua Portuguesa, como a designação de um objeto ou qualidade mediante uma palavra que designa outro objeto ou qualidade que tem com o primeiro uma relação de semelhança (por exemplo, ele tem uma vontade de ferro, para designar uma vontade forte, como o ferro). Ainda mais clara é a definição dada pelo Dicionário de la Real Academia Española: *metáfora* é a aplicação de uma palavra ou de uma expressão a um objeto ou a um conceito ao qual não se denota literalmente, com finalidade de sugerir uma comparação (com outro objeto ou conceito) e facilitar sua compreensão. Ou seja, quando queremos veicular um conceito abstrato de uma forma mais familiar e acessível freqüentemente lançamos mão de expressões metafóricas.

Na *metáfora*, o efeito principal é revelar pressuposições. Ou seja, quando alguém se expressa metaforicamente, está dizendo que pressupõe que o ouvinte conheça o campo metafórico e em função dele entenda o dado novo (BROWN, 1983). Fazemos uso dela diariamente, às vezes sem perceber, sendo parte integrante de nossa linguagem, utilizado-a como recurso estilístico, de retórica ou simplesmente para facilitar o ato de comunicação ou explicação (ERICKSON, 1995). Aristóteles afirma que "o dom de elaborar boas *metáforas* depende da capacidade de ponderar sobre semelhanças." A isto Paul Ricoeur acrescenta: "a clareza de boas *metáforas* resulta de sua capacidade de colocar frente aos olhos o sentido por ele exposto." (RICOEUR, 1975) "A *metáfora* é talvez uma das mais frutíferas potencialidades humanas. Sua força está próxima da mágica, e ele parece ser uma ferramenta de criação que Deus esqueceu dentro de cada uma de Suas criaturas quando Ele as criou." (GASSET; ORTEGA, 1925)

O uso de *metáforas* em computação pode parecer algo estranho, pois os computadores

são máquinas lógicas; sua linguagem nativa é rígida, matemática e até mesmo estranha para nossos padrões, sendo que eles não foram projetados para a complexidade da mente humana, não havendo espaço para a compreensão de entrelinhas, trocadilhos, poesia, subjetividade ou formas.

O computador, no entanto, tem a sua própria linguagem, que é bastante estranha à compreensão humana. Porém, além de ser uma ferramenta, é também um meio de comunicação e expressão, e para que haja uma comunicação efetiva homem-máquina, a linguagem computacional deve ser preferencialmente substituída pela nossa linguagem natural. Neste contexto a metáfora participa como um elemento tradutor do mundo dos computadores para algo que possamos compreender (MANDEL, 1997).

”A compreensão da metáfora, inerente às interfaces e linguagens computacionais, passou a ser usada como um instrumento para a sua melhoria, de modo que os ambientes apresentados ao usuário lhe sejam familiares e se encontrem dentro do domínio de conhecimento”. (SANTANCHÈ; TEIXEIRA, 2000)

No projeto das interfaces gráficas as metáforas desempenham um papel determinante. Foi a Xerox uma das primeiras empresas que se deu conta da importância de criar interfaces visuais, simulando alguma parte do mundo físico que fosse familiar ao maior número de pessoas. Desta forma, se convertia em tarefa fácil a compreensão pelos usuários da finalidade e uso de cada um dos elementos presentes na interface visual, como exemplo dos escritórios com seus arquivos, pastas, lápis, borrachas e outros (PREECE *et al.*, 1994).

A partir do modelo de representação desenvolvido pela Xerox, a Apple, em seus primeiros computadores Lisa e Macintosh, a chamada metáfora da mesa de trabalho ou escrivaninha (origem do termo *desktop*, em inglês), está hoje implementada em praticamente todos os ambientes operacionais. A partir deste momento, muitos projetistas passaram a adotar a metáfora como uma solução para criar interfaces dos aplicativos.

A esse respeito, Joy Mountford, pesquisador da Apple, afirma: ”Quase todos na Apple conhecem a frase: metáfora da mesa de trabalho (termo original ”desktop metaphor”), e acreditam fervorosamente que uma boa metáfora é essencial para uma interface homem-máquina fácil de utilizar”(LAUREL, 1992).

Em seu livro ”Software Amigável”, Paul Heckel chega a listar algumas vantagens do uso da metáfora. Para o autor, as metáforas tornam as coisas mais familiares, pois ”os

objetos e as regras não são coisas arbitrárias a serem memorizadas, mas têm associações com o mundo real que auxiliam o pensamento.”(HECKEL, 1993)

Inegavelmente, as interfaces gráficas, ao se tornarem amigáveis ao usuário (*user friendly*), colocaram de vez o computador na lista de eletrodomésticos básicos do dia-a-dia. Ele tornou-se “amigável” porque inteligentemente soube se aproveitar de metáforas para se fazer compreender. Ted Nelson, que trabalhou nas equipes que desenvolveram o Star, Lisa e Macintosh, afirmou que o trabalho do projetista digital deve concentrar-se 10% na concepção visual ou apresentação, 30% no projeto de interatividade e 60% na determinação de uma metáfora. “Transferindo a experiência do mundo real ao seu redor para o mundo dos computadores, os usuários contam com modelos para guiar a sua interação com computadores. É aí que entra o conceito de metáfora.”(MANDEL, 1997). Razão pela qual o emprego de metáforas no projeto visual de interfaces homem-máquina, quando é possível, constitui um importante fator de sucesso para a usabilidade dos sistemas (BENGOCHEA, 2005).

Sabe-se que o ser humano percebe o mundo por meio de um sistema sensorial que é razoavelmente bem compreendido. Quando uma interface ser humano-computador (*Human Computer Interface* - IHC) é considerada, predominam os sentidos visual, tátil e auditivo. Eles possibilitam que o usuário de um sistema baseado em computador perceba as informações, armazene-as na memória (humana) e processe-as, usando o raciocínio indutivo ou dedutivo.”(PRESSMAN, 1995)

O que está em evidência nessa pesquisa é o abismo lingüístico que existe em uma relação de comunicação mediada por computador, que tem de um lado os técnicos, engenheiros e cientistas da computação, e do outro, o usuário comum. E que tem a própria linguagem do computador de um lado e as linguagens humanas do outro. Essa situação leva a crer que a aplicação tão intensa da metáfora na computação se dá em função da necessidade de facilitar e/ou traduzir um universo tão estranho para a maior parte das pessoas pois “... metáforas servem como auxiliares ao entendimento atuando como medidores cognitivos cujos rótulos são menos técnicos que os do jargão computacional.”(ROCHA; BARANAUSKAS, 2003)

O objetivo deste trabalho não é o de abordar todas as formas de metáfora já utilizadas na computação, mas esclarecer seu uso e sua presença no projeto de programas de

computador, em que se irá apresentar as metáforas mais usuais encontradas nas interfaces dos programas da atualidade.

## 2.1 Metáfora da Mesa de Trabalho

À medida que o tempo foi passando, o crescimento e a popularização do uso de computadores por diversos perfis de usuários, com idades variadas, diversas profissões e de empresas diferenciadas, a função do computador tornou-se difícil de definir ou classificar dentro de um universo comum como, por exemplo: eletrodomésticos, meios de comunicação, máquinas de cálculo, jogos eletrônicos, instrumentos de trabalho, etc. No entanto, sem dúvida alguma, o computador se popularizou como ferramenta. Enquanto era uma máquina de cálculo científica, ele ficou restrito aos ambientes de pesquisa. Seu uso se massificou quando entrou nas empresas e nos escritórios para digitar cartas e memorandos, produzir impressos, fazer orçamentos, criar apresentações e armazenar informações. Somente depois da expansão de seu uso é que o computador invadiu o ambiente doméstico, entretendo e auxiliando nas tarefas do dia-a-dia. E se o computador se popularizou justamente no ambiente corporativo, nada mais apropriado do que colocar a mesa de trabalho (*desktop*) dentro dele. O sucesso da metáfora da mesa de trabalho se deve, em grande parte, a isso.

Quanto ao conceito de metáfora da mesa de trabalho ele foi constituído com base na crença de que usuários estão familiarizados com o ambiente da empresa, sabendo como utilizar os objetos ao seu redor como: pastas de papéis, gabinetes, telefone, bloco de notas, etc., e com isto, se sentem confortáveis com a idéia de uma mesa de trabalho de escritório como espaço de trabalho (MANDEL, 1997). Os projetistas aplicaram então esta metáfora ao mapearem as coisas que os usuários fazem no computador conforme eles fariam (no escritório), se não estivessem usando o computador.

O emprego de uma interface baseada em metáfora representa o ambiente de trabalho, onde residem projetos correntes e recursos para acessá-los. Na tela são mostradas imagens de objetos de escritório, como documentos, pastas, gavetas de arquivos, cestas. Como pequenas imagens ou ícones (SMITH *et al.*, 1982).



FIGURA 2.1 – Imagem do *Desktop* do Windows versão 3.11 em 1985 - Metáforas da Mesa de Trabalho

## 2.2 Metáfora das Pastas e Documentos

Um dos benefícios da metáfora da mesa de escritório (*desktop*) foi o de tentar eliminar o conceito de diretório, que é uma lista de nomes de arquivos ou simplesmente o fichário onde se pode encontrar um arquivo, ou seja, o diretório é um nome de certo modo distante, frio, até mesmo um pouco tecnicista. Na interface gráfica, reelaborada pela metáfora do tampo de escrivaninha, o diretório não se aplica, pois no ambiente corporativo, as pessoas guardam os seus documentos em pastas (do inglês *folder*).

As pastas surgiram com a finalidade de dar um nome mais adequado ao local onde habitualmente uma pessoa guardaria os seus documentos, sendo inspiradas na rotina do escritório da mesa de trabalho.

Portanto, mais do que renomear o conceito de "diretório", o novo termo, "pasta", reconceituou o entendimento de como os arquivos ou as informações são armazenadas e organizadas no computador, deixando de ser uma simples lista de arquivos (diretório).

A metáfora já utilizada e empregada como "diretório" pelos grupos de pesquisas, dá lugar à metáfora de "pastas" de um escritório. Neste caso, não se fala mais que um arquivo de computador está embaixo de um diretório e, sim, que um documento está dentro de uma pasta.

Outro termo empregado é o termo "arquivo", que foi dando lugar à palavra "documento", dependendo do contexto. A metáfora "documento" é mais apropriada do que

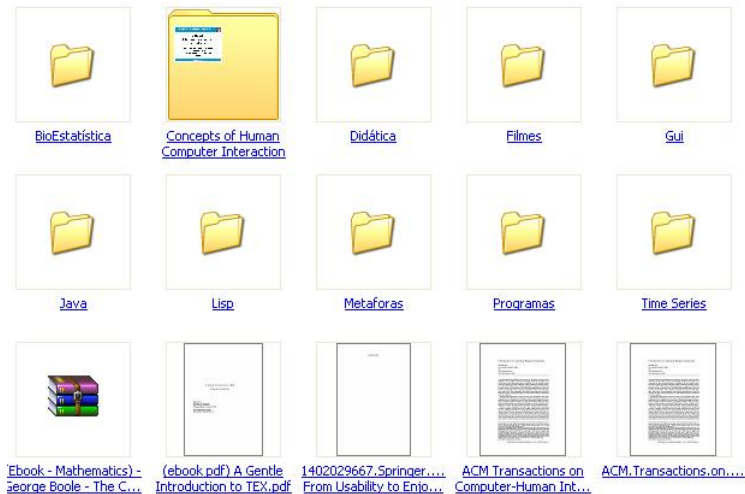


FIGURA 2.2 – Documentos e Pastas no Windows XP - Metáforas das Pastas e Documentos

”arquivo” para descrever os dados que foram salvos a partir de um processador de texto ou de uma planilha. Pois no ambiente real de trabalho em que as pessoas trabalham sentadas em frente a uma mesa, é mais comum guardar documentos em pastas do que arquivos em diretórios.

## 2.3 Metáfora dos Formulários

Os documentos, cadastros, questionários e formulários são ações e atividades familiares nos escritórios. Utilizam-se estas ações, por exemplo, quando se vai ao médico, quando se tem que preencher os formulários de imposto de renda, quando se faz uma compra na qual temos que preencher cadastros de crédito. Está presente também nas páginas em que se navega pela *Internet*, onde ler documentos ou o preenchimento de cadastros e campos é freqüente.

Deste modo, é natural que as interfaces em forma de formulários sejam populares. Quando se escreve um texto em um editor de textos ou quando se escreve um *e-mail* se está aplicando a metáfora do documento e quando se preenche campos em um programa de cadastro de clientes, contabilidade, ou até mesmo nos campos de destino e assunto das mensagens de *Internet*, está sendo aplicada a metáfora do formulário.

```
C:\>dir /w

O volume da unidade C é MS-DOS_6
O número de série do volume é 223D-81C6
Diretório de C:\

[AGENDA]      [BANNER]      [CLIPART]      [CLIPPER5]
[COREL40]      [DBASE]       [DBD]         [DELPHI]
[EXCEL]       [FLOW4]       [GAMES]       [IBLOCAL]
[INSTANT]     [LOCADORA]    [MCAFEE]     [MSOFFICE]
[POWERPNT]    [RPTSMITH]   [RS_RUN]     [SCAN]
[TURBO]       [WINDOWS]    [WINWORD]    [WINZIP]

          30 arquivo(s)             0 bytes
                                130.695.168 bytes livres

C:\>
```

FIGURA 2.3 – Tela de console do MSDOS v 6.0 - Metáfora do Console

## 2.4 Metáfora de Console

As interfaces orientadas pela metáfora do console foram as primeiras a serem utilizadas. Nos computadores de grande porte, só existia uma tela de texto e um teclado para se comunicar com os programas. A este mecanismo de comunicação com os computadores chama-se de console e tem suas vantagens: alta velocidade de processamento, utilização de uma espaço mínimo na tela do computador e rapidez de interação, principalmente para usuários experientes, já que é baseada na inserção de dados por digitação. Mas as desvantagens são evidentes. É necessário um conhecimento prévio para operá-las, principalmente se orientadas por comandos. Normalmente o usuário precisa decorar um vocabulário e uma gramática (ou seus equivalentes em computação: comandos e sintaxe) próprios para utilizar os sistemas de forma adequada. A estrutura interfacial não é visível. O nível de abstração é muito alto. E apesar da utilização de metáforas no processo de denominação de algumas instruções (nomes dos comandos), alguns comandos não fazem sentido para o usuário mais leigo, como exemplo o comando 'DIR' que significa listar os arquivos de um diretório, isto para os ambientes operacionais da Microsoft (Windows e DOS) e o mesmo ocorre com seu paralelo no mundo Unix, cujo comando é o 'ls' (FREITAS, 2005).

Este tipo de interação é mais apreciado por usuários experientes e usuários de Unix, pois vários deles aprenderam a utilizar o computador através deste tipo de interface e já têm memorizado e aprendido boa parte dos comandos que utilizam.

Esse tipo de interface está presente em todos os ambientes operacionais, mas devido

```

072006ZSEP      RETRIEVAL/GRAPHICS - REQ & SCHED      SUBSYSTEM CODE [E]
RV-607-2        ?=HELP,X=MENU,Z=EXIT JOPES OR FUNCTION CODE [G]

DATA SELECTION - OPLAN CRITERIA MENU

[E] ENTER ONE DATA RETRIEVAL OPTION (A THRU J)
  A - BY REQUIREMENT NUMBER(S)
  B - BY ORIGINATING GEO(S)
  C - BY POE GEO(S)
  D - BY POD GEO(S)
  E - BY EAD -- ENTER START DATE [N999] AND STOP DATE [C999]
  F - BY LAD -- ENTER START DATE [C000] AND STOP DATE [C009]
  G - BY FORCE MODULE(S)
  H - BY SOURCED UNIT UIC(S)
  I - BY UTC(S)
  J - ALL REQUIREMENTS
[ ] ENTER 'X' TO INCLUDE UNIT INFORMATION DATA IN RETRIEVAL
[ ] ENTER 'X' TO INCLUDE CARGO DETAIL DATA IN RETRIEVAL
[ ] ENTER 'X' TO INCLUDE CARGO EQUIPMENT QUANTITY DATA IN RETRIEVAL
^NOTE: SEE HELP SCREEN BEFORE INCLUDING CARGO EQUIPMENT QUANTITY DATA

```

FIGURA 2.4 – Menu de Tela Inteira - Metáfora do Menu

à popularização das interfaces gráficas, é necessário chamá-lo através de um comando específico, através de um ícone ou opção do menu disponíveis nas interfaces gráficas dos programas.

## 2.5 Metáfora do Menu

O primeiro avanço para tornar as interfaces mais amigáveis foi através da utilização de menus. O menu é lista de opções ou entradas postas à disposição do usuário, que aparece no vídeo de um terminal de computador com as funções que este poderá realizar por meio de um programa ou de um *software* (KOOGAN; HOUAISS, 1998). Este termo tem origem no francês (1718). Na língua portuguesa, o filólogo brasileiro Antônio de Castro Lopes criou um neologismo para substituir o menu francês pela palavra "cardápio" (1899), do latim: *chárta,ae* = papel; *carta* + *dáps,îs*: banquete oferecido aos deuses; refeição, comida (KOOGAN; HOUAISS, 1998). Entretanto, a palavra *menu* já está incorporada na linguagem computacional, em que cada item ou opção chama outro programa. Antes mesmo do aparecimento da metáfora da área de trabalho (*desktop*), muitos programas já apareciam com menus.

Ressalta-se que a metáfora do menu computacional é estabelecida pela semelhança estrutural com o cardápio de um restaurante. Um cardápio de restaurante é útil porque nos mostra o que há à disposição para comer e torna a escolha mais fácil. Da mesma forma, os menus computacionais podem ser uma ferramenta útil porque, ao apresentarem

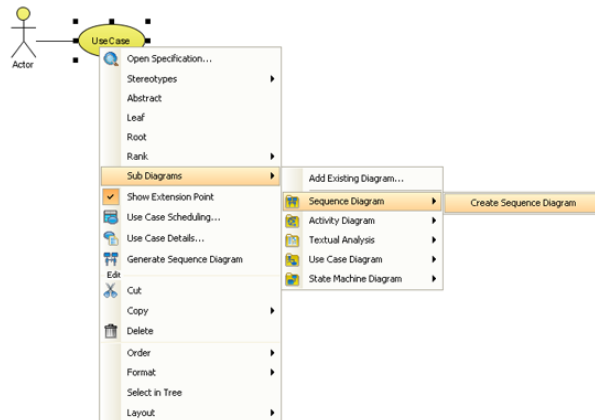


FIGURA 2.5 – Menu suspenso - Metáfora do Menu

as opções de que o usuário dispõe, eles lhe dão uma oportunidade de reconhecer o que deseja fazer (HECKEL, 1993).

Na figura 2.4 temos um exemplo de menu computacional, onde o usuário seleciona a opção desejada.

Uma vantagem na utilização dos menus é que se está familiarizado com o seu uso, por serem metáforas dos cardápios de restaurantes, cafeterias, açougues, padarias e etc. A idéia que está por trás da utilização desta metáfora é que ao saber consultar um cardápio, consegue-se também interagir com os menus do computador.

Segundo Theo Mandel (1997), os menus do computador são usados basicamente para dois propósitos. Eles permitem ao usuário navegar de um ponto para outro do sistema, ou de um menu para outro. Eles permitem também selecionar itens de uma lista. Estes itens estão usualmente relacionados a propriedades ou comportamentos de objetos que os usuários escolhem para alterar. Portanto a metáfora para os menus está fortemente relacionada com a idéia de escolha.

O nome veio, no entanto, do modelo mais antigo de menu computacional: o menu de tela inteira, no qual um quadro apresentava uma lista de itens para serem escolhidos. Quando um deles era selecionado, um programa era executado e uma tarefa era realizada. Às vezes, antes disso, se navegava de menu a menu - ou mais especificamente de tela a tela, até chegar à escolha final.

Somente mais tarde, com o lançamento do Windows 95, surgiram os menus com a sua configuração atual, nas barras de menu. Atualmente é possível encontrar ainda uma outra



FIGURA 2.6 – Exemplo de Botões



FIGURA 2.7 – Exemplo de Botões com Imagens

variação: os menus do tipo suspensos (*pop-up*), ilustrado na Figura 2.5, que aparecem quando clicamos com o botão direito do mouse.

A idéia subjacente dessas interfaces era simples: fornecer ao usuário uma série de opções e não forçá-lo a decorar, recordar e digitar uma série de comandos para executar uma tarefa específica. Esta idéia continua a fazer parte do projeto e desenvolvimento das atuais interfaces. Outro ponto importante é o de simplificar a interação entre o usuário e o computador, devendo ser clara e direcionar o usuário para uma correta utilização dos recursos do *software*, evitando excessos e confusões em sua utilização e direcionando-o a um bom fluxo de trabalho.

## 2.6 Metáfora do Painel de Controle

Tipicamente os painéis de controle que conhecemos utilizam botões para ativar ou desativar determinada função, como, por exemplo, os botões de um televisor, um rádio ou o painel de um carro que são repletos de botões, controles e mostradores. Os botões utilizados em computação são representações dos reais, portanto são virtuais. Quando se clica num desses na interface gráfica, se está executando virtualmente a ação física real, como a ação de ligar o televisor.

O aprimoramento visual dos botões virtuais é tal que existem representações para seus três estados: no seu estado normal (ou inercial), quando ele é pressionado (ou invertido) e quando está desabilitado. Em geral, todos são desenhados em três dimensões, para que a representação fique ainda mais realista.

Os sistemas operacionais atuais ainda usam mais dois tipos de botões: os botões de opção (*radio buttons*) e as caixas de opções (*check boxes*). Os primeiros lembram os

FIGURA 2.8 – Exemplo de botão de checagem (*check box*)FIGURA 2.9 – Exemplo de botão caixa de rádio (*radio box*)

orifícios dos painéis de chaves das antigas telefônicas. As caixas de opções, por outro lado, têm o formato de quadrados. E são metáforas dos campos de opção, típicas de formulários em papel e muito utilizados também nos exames vestibulares para questões de múltipla escolha. Portanto, ao clicar neles, o usuário deixa uma marca de "x". Ambos são botões de seleção. Enquanto os primeiros só aceitam uma escolha por vez, as caixas de opções são multi-selecionáveis.

## 2.7 Metáfora do Ícone

Os ícones do computador são ícones que funcionam como pontos-quentes . Ou seja, são pontos de interação, que podem ser clicados (ou duplo-clicados) para que se execute alguma ação . Assim, ícones representam diferentes tipos de objetos (programas, pastas, arquivos, unidades de discos, dados, o atalho de um arquivo) que executam diferentes ações: ao clicar (ou duplo-clicar) no ícone representativo de um programa, este é aberto; ao clicar em um ícone que representa uma pasta, esta é aberta, mostrando os arquivos que estão dentro dela; e assim por diante. Portanto, o ícone de um caderno pode ser um atalho para um editor de textos; o ícone de uma porta utiliza a imagem de uma porta não para representar uma porta, mas para representar uma ação associada: a idéia de abrir e fechar, ou seja, a metáfora está presente na idéia de que assim como fechamos uma porta; também fechamos um aplicativo. Na Figura 2.10 temos o exemplo de alguns ícones comuns nos computadores domésticos.



FIGURA 2.10 – Ícones - Metáfora do ícone

### 3 Interface Gráfica de Usuário

A primeira interface gráfica de usuário (ou GUI - *Graphical User Interface*) foi implementada pela primeira vez por Ivan Sutherland, do MIT (Massachusetts Institute of Technology) que em 1962 desenvolveu o SketchPad, um programa que permitia aos usuários criar linhas, círculos e pontos (SUTHERLAND, 1963). Mas foi no início da década de 80 que as interfaces amigáveis ao usuário começaram a povoar os monitores dos computadores pessoais, ainda em formato de texto, com simples janelas e menus (SCHNEIDERMAN, 1983). As primeiras interfaces criadas eram orientadas por linhas de comando.

Desde então tem crescido a preocupação, pelos pesquisadores e desenvolvedores de sistemas, de adequar a interface dos recursos tecnológicos ao trabalho humano. Percebe-se que os esforços voltados para esse campo têm aumentado, gradativamente, em virtude da disseminação do uso do computador pelo ser humano.

O estudo da interação Homem-Máquina é uma área do conhecimento que integra disciplinas de diferentes campos de atuação. Profissionais de computação, juntamente com psicólogos, comunicadores, *designers*, filósofos, entre outros, confirmam a necessidade de haver um maior investimento voltado para o desenvolvimento de projetos de interfaces com ou de *softwares* de alta qualidade. Por isso, os processos de elaboração e avaliação de interfaces de *softwares* vêm sendo padronizados, a título de atingir os graus de qualidade ansiados por esses profissionais e por seus usuários.

Constata-se que interfaces gráficas diferentes são utilizadas atualmente e todas elas são baseadas nos mesmos conceitos que foram implementados nos equipamentos Xerox Star e no Apple Lisa, com janelas retangulares, barra de título, botões de fechamento da janela (botões com um x no canto superior direito da janela), barras de rolagem, barras de menu no canto superior da tela, com opções de comandos em forma de texto e ícones que executam programas.

Para que a mágica da revolução digital ocorra, um computador deve também representar a si mesmo ao usuário, numa linguagem que este compreenda. (JOHNSON, 2001)

Recorde-se que em qualquer interface, gráfica ou não, o usuário valoriza a facilidade com que consegue executar as tarefas pretendidas e a comodidade ao executá-las. Por isso, as questões-chaves na implementação de uma interface são a sua finalidade e usabilidade. As escolhas e decisões a tomar, ao longo do processo de criação da interface, devem ser feitas com base na compreensão e conhecimentos dos usuários. Isso envolve levar em consideração os conhecimentos e habilidades dos usuários; saber aquilo em que as pessoas são hábeis ou não e o que elas realmente sabem usar; considerar o que poderá ajudar os usuários no modo como executam as suas tarefas; ponderar no que poderá dar qualidade às funcionalidades do sistema; ouvir o que as pessoas querem e envolvê-las no processo de criação e adotar as técnicas testadas e melhor avaliadas pelos usuários ao longo de todo o desenvolvimento da interface.

”O usuário deve sempre ser o foco central de interesse do projetista ao longo da construção e concepção da interface. O objetivo da análise e modelagem de usuários é identificar quem são os usuários e caracterizá-los, isto é, especificar quais funções exercem, quais capacidades possuem, etc”(SOUZA *et al.*, 1999).

Resumidamente, é preciso identificar as necessidades do usuário e, a partir daí, criar uma interface útil, utilizável e agradável.

Essencialmente, o processo de desenvolvimento de uma interface envolve quatro atividades básicas (PREECE *et al.*, 1994):

1. Identificar necessidades: é necessário estudar o comportamento do usuário e a forma como ele executa as tarefas, a fim de moldar a interface às suas necessidades;
2. Idealizar interfaces que respondam a essas necessidades;
3. Construir versões interativas das interfaces para que possam ser utilizadas (protótipos). Deve-se confrontar os futuros usuários com estes protótipos de forma a efetuar testes para averiguar a sua eficiência e aceitabilidade;
4. Avaliar o que foi construído ao longo do processo. Avaliar a usabilidade da interface.

Esses passos complementam-se e devem ser repetidos tantas vezes quanto o necessário.

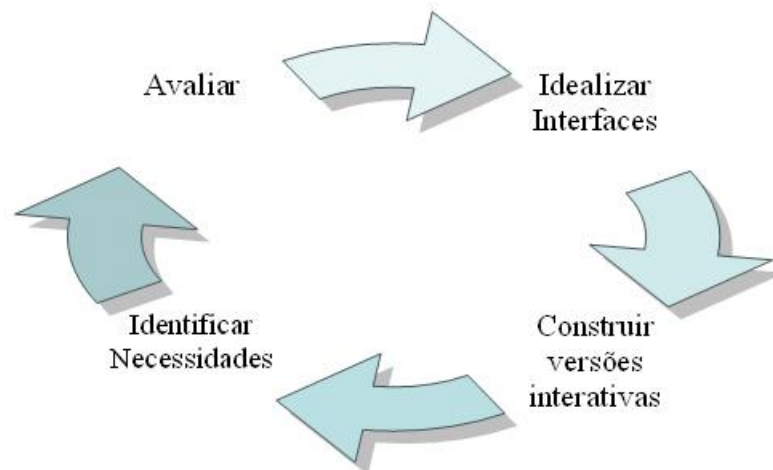


FIGURA 3.1 – Processo de Desenvolvimento de Interface

Ao avaliar o que foi construído identificam-se novas necessidades e idealizam-se mudanças a efetuar na interface ou desenvolver novas interfaces. Avaliar o que foi construído é o âmago da criação de uma interface e a melhor maneira de fazê-lo é testando o produto, testando a usabilidade da interface.

### 3.1 O que é Usabilidade?

”Usabilidade é um atributo de qualidade que avalia o quão fácil uma interface é de se usar. A palavra *usabilidade* refere-se também aos métodos de melhoramento da facilidade de utilização durante o processo de criação”(NIELSEN, 2003).

”Usabilidade é a medida de qualidade da experiência de um usuário ao interagir com um produto ou um sistema - seja um *Web site*, uma aplicação, tecnologia móvel, ou qualquer dispositivo operável por um usuário.”(USDHHS, 2007)

”Usabilidade: a extensão a que um produto pode ser usado por determinados usuários de modo a alcançar objetivos específicos com eficácia, eficiência e satisfação num determinado contexto de uso.”(ISO, 1994)

Segundo Nielsen, a usabilidade compreende cinco componentes de qualidade: facilidade de aprendizagem, facilidade de memorização, eficiência, segurança e satisfação (NIELSEN, 2003).

De acordo com Preece, a eficácia e utilidade da usabilidade definem-se em sete componentes de qualidade inerentes ao termo usabilidade (PREECE *et al.*, 1994):

- Facilidade de aprendizagem refere-se a quão facilmente a interface se aprende a usar. Ninguém gosta de gastar muito tempo para aprender a usar algo. Uma opção é começar a trabalhar imediatamente com o produto;
- Facilidade de memorização diz respeito a quão facilmente nos lembramos do modo de utilização da interface depois de a termos aprendido a usar. Se as operações a aprender são ilógicas, obscuras ou com seqüências pobres tendemos a esquecer o que fazer se usarmos poucas vezes a interface;
- Eficiência mede a rapidez com que o usuário realiza as suas tarefas. Também mede o nível de produtividade;
- Segurança envolve a proteção do usuário de condições perigosas e situações indesejáveis. Há que prevenir que o utilizador cometa erros graves reduzindo o risco de ativação de botões ou teclas erradas e dar aos usuários vários meios de recuperação caso algum erro seja cometido;
- Satisfação mede o quanto os utilizadores gostaram da interface;
- Eficácia mede a adequação da interface, se ela permite fazer bem o que é suposto;
- Utilidade refere-se à capacidade da interface apresentar as funcionalidades certas para que o usuário atinja o seu objetivo.

Preece introduz no campo da usabilidade alguns princípios de *design* (Preece, 2002):

- Visibilidade: quanto mais visíveis estiverem as funções, mais facilmente o usuário saberá o que fazer em seguida;
- *Feedback*: ação/reação. Utilizar *feedback* na forma e quantidade certa pode gerar a visibilidade necessária para a interação com o usuário;
- Restrições: dizem respeito à determinação de maneiras de restringir a interação num determinado momento. Em interfaces gráficas é usual desativar certas opções de um menu, restringindo assim o usuário às ações permitidas nesse estágio da atividade.

Uma das vantagens destas restrições é impedir que o usuário selecione uma opção incorreta, o que levará a um aumento do risco de erro;

- **Consistência:** operações semelhantes devem ter objetivos idênticos. Se numa janela um botão azul abre um documento e um vermelho o grava, na janela seguinte o botão azul pode mostrar um gráfico e o vermelho gravá-lo, por exemplo. Uma interface consistente é mais facilmente aprendida e utilizada;
- **Atribuição correta:** refere-se aos atributos de um objeto que permitem a uma pessoa saber como utilizá-lo. Um campo em branco sugere a ação "escrever aqui"; um botão com relevo traduz "clicar aqui", etc.

Por serem interativas, as interfaces gráficas adquirem uma dimensão a mais. Ted Nelson, um dos mais atentos estudiosos de sistemas interativos, "percebeu a sensação de satisfação nos usuários, quando uma interface é construída pelo que ele chamou de princípio da virtualidade - uma representação da realidade que pode ser manipulada."(SCHNEIDERMAN, 1983). Já Chris Rutkowski apresentou um conceito similar através de seu princípio da transparência: "O usuário deve concentrar o seu esforço intelectual diretamente na tarefa; a ferramenta em si deve desaparecer."(SCHNEIDERMAN, 1983). Este universo de conceitos e princípios, Ben Shneiderman chamou de "manipulação direta". A interação do receptor com a interface provoca no usuário a sensação de que ele está em contato direto e real com o ambiente representado.

O grande sucesso dos videogames está em conseguir iludir o usuário, a ponto de perder a noção de estar, na verdade, agindo sobre uma representação. O mesmo pode-se dizer dos simuladores de vôo. O usuário se sente imerso na interface. Alan Kay afirma que o interagente experimenta um estado de ilusão, de mágica (o autor estabeleceu o termo *user illusion*) (LAUREL, 1992). A interface é a parte integral da experiência do usuário.

"A sofisticação das interfaces homem-computador tem contribuído para tornar o uso das ferramentas da Informática mais amigáveis. Isto quer dizer que a alfabetização em Informática vai se tornar mais fácil graças à evolução de suas técnicas de utilização"(DINIZ, 1995). Ou seja, uma interface define o modo como o usuário interage com um programa. Por isso, deve ser fácil de usar, mas eficaz, intuitiva, eficiente. Uma boa interface complementa o potencial de um programa.

Quando questionado sobre o fator mais importante em usabilidade, Jakob Nielsen respondeu que é compreender verdadeiramente as tarefas que os usuários estão tentando realizar, porque se estiver resolvendo o problema errado você pode chegar a uma grande solução, mas não irá ajudar ninguém (OLIVER, 2003). Ou seja, uma interface que não corresponde adequadamente aos seus requisitos, não pode ser considerada uma boa interface e deve ser revista conforme descrito anteriormente.

Um bom projeto de GUI deve trazer ao usuário elementos familiares a ele na tela do computador e sua funcionalidade deve ser coerente ao seu conhecimento e reconhecida rapidamente e de forma intuitiva. Um exemplo são os programas que controlam reprodutores de música no computador com teclas semelhantes ao de um eletrodoméstico comum, como rádio ou reproduutor de discos de CD, as já conhecidas "Play", "FF", etc.

## 4 Biblioteca Proposta

Este trabalho tem como objetivo dar aos usuários e desenvolvedores, da linguagem *Objective Caml*, recursos visuais interativos (*widgets*) familiares ao usuário, utilizando a metáfora do documento/formulário e do painel de controle, explicados nos tópicos anteriores e que se considera, neste trabalho, elementos suficientes para o desenvolvimento de aplicações poderosas e completas, sem no entanto esgotar todas as opções de criação de recursos visuais.

Todos os programas e rotinas desta pesquisa foram implementados em ML, mais especificamente na linguagem Objective Caml (OCaml) desenvolvida e distribuída pela INRIA (Instituto Nacional de Pesquisa em Ciência da Computação - Institut National de Recherche en Informatique et en Automatique) desde 1985 (LEROY, 2002; Remy et al, 1997).

OCaml é um dialeto de ML (Meta Linguagem), que é um derivado da linguagem ML clássica desenvolvida por Robin Milner, em 1975, para o provador de teoremas LCF (Lógica de Funções Computacionais - Logic of Computable Functions) (Gordon *et al* 1979, (SMITH *et al.*, 1982)). Demais informações sobre a Linguagem OCaml, incluindo seu pacote de instalação, código fonte, manuais, sites de colaboração, notícias, novidades, *sites* com bibliotecas e outros recursos podem ser encontrados e acessados pelo endereço de *Internet* [www.ocaml.org](http://www.ocaml.org).

Algumas das motivações para a escolha desta linguagem foram ((GRINGS, 2006); (SOARES, 2007)):

- Programação multiparadigma. Embora a OCaml seja originalmente pertencente à família das linguagens funcionais, ela também permite os estilos de programação imperativo e orientado a objetos, todos de maneira integrada no próprio projeto da linguagem;

- Gerência automática de memória. Todas as alocações e liberação de memória estão a cargo do próprio compilador, o que torna o programa mais eficiente na utilização da memória;
- Portabilidade. Os programas escritos nesta linguagem podem ser compilados, praticamente sem alteração, em Windows e em diferentes versões do Unix (incluindo Linux e Mac OS X), sendo que durante o desenvolvimento deste trabalho se realizaram testes bem sucedidos nos ambientes Windows e Linux (distribuição Ubuntu versão 7.04) .
- Geração de código eficiente. OCaml possui um gerador de código nativo rápido e eficiente. A qualidade do código gerado é comparável com aqueles gerados por compiladores C e, algumas vezes, até melhor;
- A linguagem possui um interpretador. Além do compilador, OCaml também possui um interpretador, o que acelera e facilita a depuração de erros e o ciclo de desenvolvimento de programas;
- Grande número de bibliotecas. OCaml possui várias bibliotecas construídas na própria linguagem para um grande número de aplicações, indo desde estruturas de dados até rotinas de cálculo numérico. Além disto, devido à facilidade de integração de código escritos em outras linguagens, como no caso da linguagem C, o sistema permite que virtualmente todas as bibliotecas escritas nestas linguagens sejam facilmente incorporadas, o que aumenta em muito a disponibilidade de bibliotecas para OCaml;
- Comunidade ativa de usuários. Seu desenvolvedor, INRIA, incentiva e mantém fóruns e comunidades interligadas pela *Internet*, onde uma comunidade ativa de usuários espalhados pelo mundo interage e colabora. Assim, a resolução de eventuais problemas e dúvidas sobre a linguagem ou algum ponto de implementação é grandemente facilitada;
- Boa documentação. Os usuários e implementadores da linguagem escrevem boa documentação, prontamente acessível pelo *site* da linguagem na *internet* e através de bons livros disponíveis gratuitamente na *Internet*, bem como alguns bons títulos pagos.

A biblioteca e produto resultado deste trabalho foi batizada de LEMAC, cujo nome tem origem na fusão dos nomes 'Ocaml' e 'EMACs', sendo o primeiro nome relacionado à linguagem sobre a qual foi desenvolvido o trabalho e o segundo nome, o editor de textos em torno do qual baseia-se parte das funcionalidades do próprio LEMAC.

O LEMAC então é uma biblioteca de rotinas que auxilia no desenvolvimento de GUI na linguagem Ocaml. Esta linguagem oferece quatro objetos básicos: botões com texto, botões com imagens (metáfora do painel de controle), editor de textos e editor de textos com campos (metáfora do formulário). Os editores desta biblioteca possuem controles de teclas semelhantes ao editor EMACS, vistos a seguir:

Movimento do cursor:

<CTRL> f - move o cursor para frente;  
<CTRL> b - move o cursor para trás;  
<CTRL> n - move o cursor uma linha para baixo, mantendo-o na mesma coluna ou na última coluna da linha se esta for mais curta;  
<CTRL> p - move o cursor uma linha para cima, mantendo-o na mesma coluna ou na última coluna da linha se esta for mais curta;  
<CTRL> a - move o cursor para o início da linha;  
<CTRL> e - move o cursor para o fim da linha;  
<ESC> a - move o cursor para o início do texto;  
<ESC> e - move o cursor para o fim do texto;  
<Clique do *mouse*> - move para o local onde o mouse está apontando, se dentro do editor; se na clicar na aba, irá ativar o editor correspondente ao nome da aba.

Comandos com marcação

<ESC> <ESPAÇO> - marca o fim do texto a ser marcado;  
<CTRL> w - apaga o texto do ponto marcado até o ponto do cursor, sendo que o texto é inserido na fila de memória (*buffer*) e pode ser utilizado posteriormente;  
<ESC> w - copia o texto para a fila de memória;  
<CTRL> y - insere o conteúdo da fila de memória no texto;

### Comandos de edição

<ENTER> - insere uma quebra de linha no texto;

<CTRL> j - insere espaços em branco até o fim da linha atual;

<BACKSPACE> - volta um caracter para trás e elimina o caracter; entretanto pode-se usar o controle <CTRL> q para desativar a eliminação do caracter;

<CTRL> q - ativa e desativa a eliminação do caracter pelo controle de edição do <BACKSPACE>

<CTRL> k - apaga a linha do ponto atual até o fim da linha (quebra de linha);

<CTRL> s - pesquisa por um texto à frente;

<CTRL> r - pesquisa por um texto para trás;

<CTRL> x <CTRL> s - salva o texto em arquivo com o mesmo nome na aba no editor;

<CTRL> x <CTRL> f - lê o texto em arquivo com o nome fornecido pelo usuário;

A listagem com o código-fonte completo da biblioteca LEMAC está contida no Anexo I deste texto. Tanto este código fonte como vários outros programas de exemplo podem ser obtidos através do endereço de *Internet* onde este trabalho foi publicado (<http://lemac.sourceforge.net/>) e onde sempre a última versão estará disponível.

A seguir será descrito como utilizar os controles da biblioteca no desenvolvimento das aplicações em OCaml.

## 4.1 Criando botões

Para exemplificar como se cria um botão utilizando o LEMAC, usar-se-á o programa exemplo abaixo:

```
1 let ola () =  
2   Wid.set_contents "Ola_Mundo" "Exemplo" ;;  
3 let main () =  
4   _____Wid.open_gr "Exemplo_de_GUI";  
5   _____let editor = Wid.mk_edt ~txt:"Exemplo_de_programa." "Exemplo" and
```

```

6          saida = Wid.exit_button() and
7          ola = Wid.button 0 (ola) "Ola" in
8  _____while true do
9          Wid.refresh ();
10         editor.Wid.process_event ();
11         ola.Wid.process_event ();
12  _____ saida.Wid.process_event ()
13  _____done;
14  _____Wid.close_gr ();;
15 main ();;

```

#### Listagem 1

A figura abaixo mostra a tela resultante do programa resultante pela listagem acima que apresenta dois objetos da biblioteca LEMAC, um botão de saída e um editor.

Para criar um botão, a sintaxe usada está descrita abaixo:  
`Wid.button < Índice> < Função> < Rótulo>`

Tem-se que:

*Índice* - é o índice da seqüência de apresentação do botão na tela;

*Função* - é a função que será executada;

*Rótulo* - é o texto que será apresentado dentro da caixa do botão.

A linha 7 da listagem x cria um botão

```
> bt_ola = Wid.button 0 (ola) "Ola!"
```

Este trecho de código cria o botão, chamando o método "Wid.button". O botão criado com o índice 0 ao ser clicado, aciona a função 'ola'; o botão será preenchido pelo texto 'Ola'.

Como exemplo, para se fazer algo semelhante utilizando-se de outras bibliotecas de interface gráfica de usuário, o código é mais extenso ou complexo. Como exemplo lista-se abaixo um trecho de código que cria uma janela com o texto "Alo Mundo" e um botão utilizando a biblioteca GTK e a linguagem C++:

```

1  /* Exemplo de interface utilizando GTK - Alo Mundo 2 - alo2.c */
2  #include <gtk/gtk.h>

```



FIGURA 4.1 – Programa exemplo gerado pela Listagem 1

```
3 void sair(GtkWidget *w, gpointer p)
4 {
5     gtk_main_quit();
6 }
7 void clique(GtkWidget *w, gpointer p)
8 {
9     g_print("O_botao_foi_clicado\n");
10 }
11 int main(int argc, char **argv)
12 {
13     GtkWidget *janela, *botao;
14     gtk_init(&argc, &argv);
15     janela = gtk_window_new(GTK_WINDOW_TOPLEVEL);
16     gtk_window_set_title(GTK_WINDOW(janela), "Alo_Mundo");
17     gtk_signal_connect(GTK_OBJECT(janela), "destroy", GTK_SIGNAL_FUNC(sair),
18         NULL);
19     botao = gtk_button_new_with_label("Clique_aqui");
20     gtk_container_add(GTK_CONTAINER(janela), botao);
21     gtk_signal_connect(GTK_OBJECT(botao), "clicked", GTK_SIGNAL_FUNC(clique),
22         NULL);
23     gtk_widget_show(botao);
24     gtk_widget_show(janela);

26     gtk_main();
27     return 0;
```

```

28 }
29 /* Fim do Exemplo */

```

Como se pode perceber o código é mais extenso e com muitos detalhes a se gerenciar, mas também há outros exemplos com código mais compacto, como no caso do Fox toolkit, como visto abaixo:

```

1 #include "fx.h"

3 int main(int argc, char *argv[]) {
4     FXApp application("Alo_Mundo", "Fox_Teste");
5     application.init(argc, argv);
6     FXMainWindow *main=new FXMainWindow(&application, "Alo", NULL, NULL,
7     DECOR_ALL);
8     new FXButton(main, "&Alo_Mundo!", NULL, &application, FXApp::ID_QUIT);
9     application.create();
10    main->show(PLACEMENT_SCREEN);
11    return application.run();
12 }

```

O código acima é mais compacto que o exemplo anterior, mas pode-se notar que sua legibilidade e complexidade são maiores.

## 4.2 Criando um Editor

Para se criar um editor padrão EMACS, a sintaxe da função é a seguinte:

```
Wid.mk_edt [~txt:<texto>] [~fn:<funcao>] <rótulo>
```

Sendo que

*Texto* - é o texto inicial do editor, este parâmetro é opcional;

*Funcao* - é a função que será executada ao clicar no botão 'x' dentro do editor, também é opcional;

*Rótulo* - é o nome do editor e que figura na aba acima do editor.

Na linha 5 da listagem 1, é criado um editor com o texto inicial definido e cujo rótulo é "Exemplo"

```
> Wid.mk_edt ~txt:"Exemplo de programa." "Exemplo"
```

### 4.3 Ativando os Objetos

Para ativar as funções internas dos objetos, é necessário chamar o método 'Wid.process\_event' que gerencia os eventos inerentes a cada objeto. O trecho de código abaixo, retirado da Listagem 1, demonstra uma forma de se executar este processo.

```
1 _____while true do
2           Wid.refresh ();
3           editor.Wid.process_event ();
4           ola.Wid.process_event ();
5 _____saida.Wid.process_event ()
6 _____done;
```

Como descrito acima, não é complicado para um desenvolvedor utilizar as rotinas propostas pelo LEMAC, ficando a maior parte do foco do trabalho na aplicação em si e não na interface.

### 4.4 Criando Botões Gráficos

Aproveitando o código do programa "nw.ml" (Anexo 2), incluído no pacote de distribuição do LEMAC, pode-se exemplificar a função de criação de um botão gráfico. Na linha 70 tem-se a seguinte chamada de função:

```
> Wid.gbutton 0 "um.nss" "tres.nss" (upc "Editor 1") "Uppercase"
```

Essa função cria um botão gráfico com índice 0, sendo que as imagens do botão são

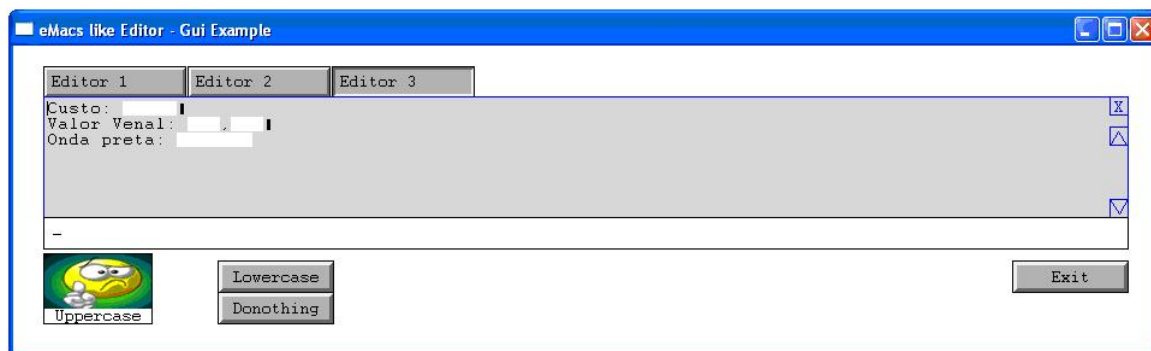


FIGURA 4.2 – Imagem do formulário gerado a partir do programa nw.ml.

dadas pelo conteúdo dos arquivos 'um.nss' e 'tres.nss', com execução da função '(upc "Editor 1")' se clicado e com rótulo 'Uppercase'.

Os arquivos gráficos foram gerados a partir de imagens do tipo *png* e transformados para o formato *nss* utilizando o aplicativo "marshall.exe", cujo programa código fonte foi fornecido juntamente com o pacote do LEMAC para possibilitar, a qualquer usuário, converter essas imagens.

## 4.5 Criando Formulários

A criação de formulários é semelhante à criação dos editores, com a substituição do texto inicial pela máscara de entrada, demonstrado através da linha retirada do código do programa "nw.ml" (anexo 2) mostrada a seguir

```
> Wid.mk_entry ~txt:msk ~fn:processa "Editor 3",
```

A mascara de entrada para o código do programa foi definida pelo trecho abaixo:

```
let msk= Wid.the_mask ["Custo: #####\013Valor Venal: ###,###";  
"Onda preta: ##### "]
```

Cada caracter '#' representa um espaço para entrada de dados e os outros caracteres são apresentados como texto normal.

A ativação desses dois últimos controles é idêntica ao descrito anteriormente, não sendo necessário repetir esta explicação.

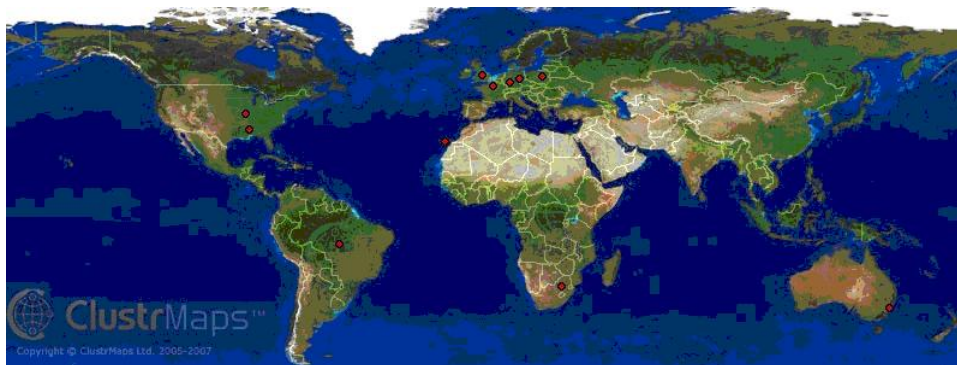


FIGURA 4.3 – Imagem da localização geográfica dos usuários que acessaram o *site* da biblioteca LEMAC através do *site* Sourceforge.net, obtido em 23 de agosto de 2007.

## 4.6 Publicação e Distribuição

O código da Biblioteca LEMAC foi publicado no <http://www.sourceforge.net/>, um famoso *site* de distribuição e publicação de sistemas com código aberto (*open source*), onde foi disponibilizado um manual sobre como utilizar as rotinas da biblioteca, o código-fonte completo e programas-exemplos.

Através deste *site*, constatou-se que ele já foi pelo menos acessado por vários usuários e desenvolvedores pelo mundo afora. Foi instalado um aditivo na página com o manual do sistema e por ela se pode inspecionar os locais onde os usuários a acessaram.

Foi registrado pelo *site* de distribuição da biblioteca que em junho foram realizados 177 *downloads* desse sistema por usuários em todo mundo.

## 4.7 Considerações Finais

Com a utilização das rotinas descritas nos itens anteriores, demonstrou-se como utilizar todas as funcionalidades da Biblioteca LEMAC. Este conjunto de rotinas permite aos desenvolvedores construir interfaces gráficas de usuário com poucas linhas de código em OCaml, de forma simples e objetiva.

## 5 Considerações Finais e Conclusões

A indústria de desenvolvimento de sistemas (*software*) vem ao longo dos anos aumentando a funcionalidade das aplicações, em uma tentativa de satisfazer parte dos anseios de grande número de usuários possíveis. Entretanto, essa solução não apenas não consegue atingir totalmente a meta, como ainda traz consigo outros problemas tão graves quanto (ou mais do que) a falta de funcionalidade. Este assunto se constitui de um amplo campo de debates, discussões e pesquisas e está sujeito a modismos e a subjetividade dos usuários.

Como proposto no objetivo deste trabalho, foi desenvolvida uma biblioteca, denominada LEMAC, para a construção de interfaces gráficas para usuário (GUI) em OCaml, que disponibiliza quatro componentes visuais para o desenvolvedor/usuário, que são editores de texto, editores de campos, botões com texto e botões com ícones; o suficiente para desenvolvedores e usuários interagirem eficientemente com os sistemas, mas sem esgotar as possibilidades de futuros acréscimos de novos componentes visuais.

O trabalho desenvolvido contemplou os seguintes objetivos específicos:

- Ser compacta: depois de compilado, o espaço em disco gasto apenas com a biblioteca será de aproximadamente 400Kbytes, já estando embutida no programa executável, como pode ser comprovado através dos vários programas de exemplo que a acompanham. Utilizando outras bibliotecas, há a necessidade de instalá-las separadamente, e estas instalações podem chegar a consumir mais de 200 Mbytes, como descrito no tópico "Instalação de Bibliotecas";
- Ser portátil (multiplataforma): por ser o compilador OCaml disponível nas plataformas Linux, Windows e Mac OS, foi possível compilar e testar programas utilizando a ferramenta desenvolvida neste trabalho nos ambientes Windows e Linux

funcionando adequadamente, como esperado;

- Ser fácil de programar: justamente por se utilizar de poucas linhas de código para utilizar os recursos da biblioteca, ela se mostrou de fácil programação, o que difere de outras bibliotecas que necessitam de muitas (mais de 10) linhas de código para executar funções semelhantes;
- Ser fácil de aprender: programadores novatos, mesmo com pouco conhecimento da arte de programar demonstraram facilidade para o aprendizado e utilização da biblioteca;
- Ser fácil de ser utilizada: programas construídos com esta biblioteca podem utilizar os controles visuais de botões, editores de texto e editores de campo, o que já é muito utilizado pelos usuário, tornando sua utilização natural para quem já os conhece.

Como visto no objetivo deste trabalho, a necessidade de bibliotecas de construção de interfaces gráficas, foi suprida no caso do OCaml com a Biblioteca LEMAC.

Este trabalho foi também focado na metáfora do documento/formulários que é vastamente utilizada no mundo real, aprendida e usada desde os bancos da escola por meio dos simples cadernos, blocos de anotação e livros didáticos fortemente representados no mundo virtual da *Internet*, em seus vários casos de formulários que freqüentemente se é obrigado a preencher.

Ressalte-se que os vários exemplos disponíveis de outros ambientes gráficos possuem características pouco recomendadas à boa prática de programação, por serem lentos e grandes consumidores de recursos da máquina, em razão do que se ateve a presente pesquisa em gerar um exemplo de ferramenta rápida, prática e econômica no consumo de recursos computacionais.

A partir disso construiu-se essa ferramenta focada nesses objetivos, modelada com a concepção de metáforas do documento/formulário, contudo dotada de excepcional rapidez, simplicidade, praticidade e portabilidade, possibilitando ao desenvolvedor gerar aplicações em menor tempo.

Com isso as aplicações, que utilizam essa ferramenta como interface padrão, tornam-se passíveis de portabilidade sem a necessidade de gerar códigos específicos para tal ou qual

plataforma, com a simplificação da tarefa do programador e incremento de sua produtividade.

Essa biblioteca foi desenvolvida para rodar em OCaml que, por basear-se na lógica matemática, apresenta a peculiaridade de manter-se atual, mesmo que a respectiva versão da linguagem se torne obsoleta. Para tanto, um mínimo de adaptação torna-a operacional às novas linguagens sucessoras.

## 5.1 Futuras Contribuições

A ciência, como um processo contínuo e interminável, vale-se das constantes conquistas e desenvolvimentos para o seu dever. Com relação à presente dissertação de Mestrado constituído pela temática da construção de Biblioteca de Interface Gráfica para a linguagem OCaml simples e prática, foi possível alicerçar significativa contribuição para profissionais de sistemas desta linguagem.

Pelo fato de alcançar tais méritos, os esforços não findam e apontam para novas perspectivas inovadoras, onde se possa aperfeiçoar e simplificar os resultados obtidos com vistas à popularização e usabilidade do instrumental em questão.

A propósito sugerem-se linhas de continuidade nas diretrizes a seguir:

- Manter, atualizar, simplificar e tornar mais eficiente o código da biblioteca já publicada;
- Realizar uma avaliação formal da interface com base na norma ISONORM 9241-10 Usabilidade (conjunto de normas que permite avaliar a capacidade de um sistema interativo oferecer ao usuário a possibilidade de realizar tarefas de maneira eficaz e agradável (Prungmper, 1999) e
- Desenvolver bibliotecas similares ao LEMAC para outras linguagens que ainda careçam deste recurso, tal como Mercury.

Tais sugestões certamente viabilizam novas concepções e iniciativas em vários campos de pesquisa pela utilização das ferramentas já consolidadas e aqui apresentadas, como

---

uma contribuição estimuladora da produtividade para a comunidade de desenvolvedores de linguagens funcionais.



# Referências Bibliográficas

APIKI, S. **Paths to Plataform Independence**. [s.n.], 1994. Disponível em: <<http://www.byte.com/art/9401/sec9/art1.htm>>. Acesso em: 20 de jun. de 2007.

BARRIER, T. **Human Computer Interaction Development**. USA: IRM Press, 2002.

BENGOCHEA, M. n. P. L. Sistemas de visualización para bibliotecas digitales. **Revista Española de Documentación Científica**, v. 28, n. 3, p. 273–292, 2005.

BRAGA, A. S. **As origens do design e sua influência na produção da hipermídia**. Dissertação (Mestrado) — Pontifícia Universidade Católica de São Paulo, São Paulo, 2004.

BREY, P. **The Epistemology and Ontology of Human-Computer Interaction**. [S.l.]: Minds and Machines, 2005.

BROWN, G. Y. G. **Discourse Analysis**. Cambridge: Cambridge University Press, 1983.

BUSH, V. As we may think. **The Atlantic Monthly**, 1945. Disponível em: <<http://ccat.sas.upenn.edu/jod/texts/vannevar.bush.html> , <http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>>. Acesso em: 23 de jun. de 2007.

CABRAL, R. de M. B. **Comunicação Homem-Computador**. Porto Alegre: [s.n.], 2007. Disponível em: <<http://www.inf.ufrgs.br/~cabral/09.Interf.Graf.Mai.2007.ppt>>. Acesso em: 01 de set. de 2007.

CARROL, J. M.; MARK, R. L.; KELLOG, W. A. **Interface Metaphors and User Interface Design**: In handbook of human-computer interaction. Amsterdam: North-Holland, 1988.

CHELARU, A. **"Feel-at-home" design for application interfaces**: Creating mirror-interfaces for the users. [s.n.], 2007. Disponível em: <[http://courses.interaction-ivrea.it/papers/papers/chelaru\\_FeelAtHomeDesign.pdf](http://courses.interaction-ivrea.it/papers/papers/chelaru_FeelAtHomeDesign.pdf)>. Acesso em: 3 de jun. de 2007.

COMPUTER, A. **Human Interface Guidelines**: The apple desktop interface. [S.l.]: Addison-Wesley, 1989.

DINIZ, E. H. O hipertexto e as interfaces homem-computador: construindo uma linguagem da informática. **Revista de Educação e Informática São Paulo**, Fundação para o Desenvolvimento da Educação, v. 5, n. 11, Dezembro 1995.

- DONDIS, D. A. **Sintaxe da Linguagem Visual**. São Paulo: Martins Fontes, 1991.
- ECO, U. Os códigos visuais. **A Estrutura Ausente: Introdução à Pesquisa Semiológica**, Perspectiva, p. 97–121, 1971.
- ERICKSON, T. D. Working with interface metaphors. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 147–151, 1995.
- FREITAS, D. S. **Utilização do LVM como Facilitador de Gerência de Partições no Linux**. Lavras: [s.n.], 2005. 40 p.
- GASSET, J.; ORTEGA. **A desumanização da arte**. São Paulo: Cortez, 1925.
- GRINGS, A. **Regressão simbólica via Programação Genética**: um estudo de caso com modelagem geofísica. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Faculdade de Computação, fevereiro 2006.
- HECKEL, P. **Software Amigável**: técnicas de projeto de software para uma melhor interface com o usuário. Rio de Janeiro: Editora Campus, 1993. 155-160 p.
- INTERNATIONAL ORGANISATION FOR STANDARDISATION - DRAFT INTERNATIONAL STANDARD. **ISO DIS 9241-11**. [S.l.], 1994. Disponível em: <<http://www.usability.ru/sources/iso9241-11.htm>>. Acesso em: 15 de jun. de 2007.
- JOHNSON, S. **Cultura da Interface**. Rio de Janeiro: Jorge Zahar Editor, 2001.
- KOOGAN, A.; HOUAISS, A. (Ed.). **Enciclopédia e dicionário digital 98**. Direção geral de André Koogan Breikman. São Paulo: Delta: Estadão, 1998. 5 CD-ROM. Produzida por Videolar Multimídia.
- LAUREL, B. **The art of human-computer interface design**. [S.l.]: Apple Computer, Addison-Wesley Publishing Company, 1992. 27 p.
- LEÃO, L. **O Labirinto da Hipermídia**. São Paulo: Iluminuras, 1999.
- MANDEL, T. **The Elements of User Interface Design**. Nova York: Wiley Computer Publishing, 1997. 25 p.
- MCKAY, R. **Plataform Independent FAQ**. [s.n.], 1997. Disponível em: <<http://www.zeta.org.au/~rosko/pigui.htm>>. Acesso em: 2 de jun. de 2007.
- NIELSEN, J. **Usability 101: Fundamentals and Definitions**. [s.n.], 2003. Disponível em: <<http://www.useit.com/alertbox/20030825.html>>. Acesso em: 8 de jun. de 2007.
- OLIVER, D. Jakob nielsen told me to do it. **.net**, n. 93, Fevereiro 2003.
- PLAZA, J. G. **A imagem Digital**: Crise dos sistemas de representação. Dissertação (Mestrado) — Escola de Comunicações e Artes da Universidade de São Paulo, São Paulo, 1991.
- PREECE, J.; ROGERS, Y.; SHARP, H.; DAVID, B.; HOLLAND, S.; CAREY, T. **Human-Computer Interaction**. [S.l.]: Addison-Wesley Publishing Company, 1994.
- PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Markron Books, 1995.

RICOEUR, P. **A Metafora Viva**. Porto: RES, 1975. 47 p.

ROCHA, H. V.; BARANAUSKAS, M. C. C. **Design e Avaliação de Interfaces Humano-Computador**. Campinas: NIED/UNICAMP, 2003.

SANTANCHÈ, A.; TEIXEIRA, C. A. C. Integrando instrucionismo e construcionismo em aplicações educacionais através do casa mágica. XIX Congresso da SBC, 2000. Disponível em: <<http://www.geocities.com/santanche/publicado/WIE99.pdf>>. Acesso em: 15 de jun. de 2007.

SCHNEIDERMAN, B. **Direct manipulation**: A step beyond programming languages. [S.l.]: IEEE Computer, 1983. 57-69 p.

SMITH, D.; IRBY, C.; KIMBALL, R.; VERPLANK, W.; HARSLEM, E. **The Star User Interface**: An overview. [S.l.]: AFIPS National Computer Conference, 1982. 512-528 p.

SOARES, A. S. **Aproximação de nuvens de pontos de dados por meio de superfícies de Bézier**. Tese (Doutorado) — Universidade Federal de Uberlândia, Uberlândia, 2007.

CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 2., 1999, Rio de Janeiro. **Projeto de Interfaces de Usuário: Perspectivas Cognitivas e Semióticas. Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação**. Rio de Janeiro: Edições Entrelugar, 1999.

SUTHERLAND, I. E. **A man-machine graphical communication system**. Tese (Doutorado) — University of Cambridge - Computer Laboratory Sketchpad, Inglaterra, 1963.

TOGNAZZINI, B. **Tog on Interface**: Reading. [S.l.]: Addison-Wesley, 1992.

U. S. DEPARTMENT OF HEALTH AND HUMAN SERVICES. **Usability Basics**. [S.l.], 2007. Disponível em: <<http://www.usability.gov/basics/index.html>>. Acesso em: 15 de jun. de 2007.

# Anexo 1

Listagem da Biblioteca Lemac:

```
1 (* Type declaration section *)

3 (* Rectangle type definition (x,y) -
4    position on the screen (w,h)=(width,height) of the rectangle *)
5    type box_config = { x:int; y:int; w:int; h:int };;

7 (* eMac's like editor object type definition *)
8 type txt = {
9    (* Text of the Editor *)
10   _____mutable tx: string;
11   (* Position of the line and cursor *)
12   _____mutable cpos:(int*int);
13   (* Position of the cursor in the current line *)
14   _____mutable stx:int;
15   (* Position of the first position to display in the window *)
16   _____mutable sty:int;
17   (* Used to mark a position in the text *)
18   _____mutable mark:int;
19   _____mutable mask: string };;

21 (* Image or colour on the button *)
22 type widg= IMG of Graphics.image | CLR of Graphics.color;;

24 (* Process definition *)
25 type widget =
```

```
26 {_____process_event : (unit -> unit);
27 _____st: (txt -> widg -> bool -> unit);
28 _____draw_self : (unit -> unit) }

30 (* Structure to storage all local variables needed in the
31 text editor *)
32 type edit_state = {
33 (* Backup stx used some times *)
34 _____mutable bkpx:int;
35 (* name of the Editor *)
36 _____mutable nm:string;
37 (* Used to activate or deactivate the erase of the caractere
38    on backspace key *)
39 _____mutable bkerase:bool;
40 (* buffer to safe the last test *)
41 _____mutable ctx_save:string;
42 (* buffer to safe the last mask *)
43 _____mutable cmk_save:string;
44 (*    *)
45 _____mutable sty_stk: int list;
46 (*    *)
47 _____mutable insertions:bool;
48 (* All main variables of the Editor *)
49 _____mutable etx:txt;
50 (* Used to storage a virtual key *)
51 _____mutable virtual_char:int}

53 (* structure to store all global variables for all objects *)
54 type glb= {
55 (* Define the number of lines showed in the window *)
56 _____
57    mutable lines_per_page:int;
58 (* Define the number of columns showed in the window *)
```

```
58 _____
    mutable chars_per_line:int;
59 (* Mouse position when an event occurs *)
60 _____
    mutable mouse_position:(int*int);
61 (* Yank definition its the list buffer that storage parts of the
62     text on memory *)
63 _____
    mutable yank:(string list*string list);
64 (* list of componets to rebuild its graphics elements *)
65 _____
    mutable refresh_list:widget list;
66 (* The last position find for any search done in editors *)
67 _____
    mutable search_pos:int;
68 (* Control a virtual keypress *)
69 _____
    mutable virtual_keys:int list;
70 _____ mutable mrk:int };;

73 (* Main variables definitions *)

75 (* Global variables initialization *)
76 let glb= {_____lines_per_page= 5;
77 _____chars_per_line= 10;
78 _____mouse_position= (0,0);
79 _____yank=([], []);
80 _____refresh_list= [];
81 _____search_pos= 0;
82 _____virtual_keys=[];
83 _____mrk=(-1)};;

85 (* Text editor controls *)
```

```
86 (* The number of Editors created *)
87 let ____top= ref 0 and
88 (* setup to control only 5 editors *)
89 (* Stack to storage the name of the editor *)
90 ____pilha= Array.make 5 "" and
91 (* The structure to storage the contents of the Editor *)
92 ____oldStuff= Array.make 5 "" and
93 (* The structure to storage the contents of the mask for all Editors *)
94 ____mskStuff= Array.make 5 "" and
95 (* create positions in the memory for 5 editors *)
96 ____eds= Array.make 5 { process_event= (fun () -> ());
97 _____
98 st= (fun x y z -> ());
99 _____
100 draw_self=(fun () -> ())} and
101 (* border size on the main window *)
102 ____winHeight=5;;

103 (* Define grey levels *)
104 let set_gray x = (Graphics.rgb x x x);;
105 (* Define several tones of grey and white used in Editors and buttons *)
106 let ____gray1 ____= set_gray 100 and
107 ____gray2____= set_gray 170 and
108 ____gray3____= set_gray 240 and
109 ____background_mask____= set_gray 215 and
110 ____background_edt____= set_gray 200;;

111 let last_sz= ref (0, 0);;

112
113 let set_siz ()= last_sz := (Graphics.size_x(), Graphics.size_y());;

114
115
116 (* Routines *)
```

```
118 (* get the top of the circular list yank-buffer *)
119 let gettop ()=
120   try
121     match glb.yank with
122       (ys, []) -> glb.yank <- ([], List.rev ys); List.hd ys
123     | (ys, (x::xs)) -> x
124   with
125     any -> "";;
126 (* circle the list yank-buffer *)
127 let circula ()=
128   try
129     match glb.yank with
130       ([], [x]) -> ()
131     | (ys, [x]) -> glb.yank <- ([x], List.rev ys)
132     | (ys, (x::xs)) -> glb.yank <- ((x::ys), xs)

134     | (ys, []) -> glb.yank <- ([], List.rev ys)
135   with
136     any -> ();;
137 (* push a string on the list yank-buffer *)
138 let push_circular x =
139   let (xs, ys)= glb.yank in
140   glb.yank <- (xs, x::ys);;

142 (* constructor of the rectangle *)
143 let mkrec ax ay wx wy= {x= ax; y=ay; w=wx; h=wy};;
144 (* constructor of the eMac like editor *)

146 let mask_fun s i=
147   let last_line= String.length s - 1 in
148   if i <0 || i > last_line then false
149   else ( s.[i]= '#' );;
```

```
151 let createText xs = {cpos = (0, 0);  
    tx = xs; stx=0; sty=0; mark=0; mask= "" };;  
  
154 let createMaskText xs msk = {cpos = (0, 0);  
    tx =xs ; stx=0; sty=0; mark=0;  
155                                mask=msk };;  
  
157 (* integer Stack controls *)  
158 let clear_stack stack = stack := [] and  
159     empty_stack stack = match stack.contents with  
160     _____[] -> true  
161     _____| _ -> false  
162 and push stack data =  
163     _____stack:=data::stack.contents and  
164     pop stack = match stack.contents with  
165     _____[] -> 0  
166     _____| (head::tail) -> stack:=tail; head ;;  
  
168 (* invert the index of a string *)  
169 let gch s i =  
170     let sz= String.length s - 1 in  
171         s.[sz - i];;  
  
173 (* search string using Boyer/More algoritm *)  
174 let search pat text index=  
175 (* computation of the shift-table - phase 1 *)  
176     _____let m=String.length pat and  
177     _____n=String.length text in  
178     _____let rborder=Array.create (m+1) 0 and  
179     _____d = Array.create (m+1) 1 and  
180     _____last = Array.create 256 0 in  
181     _____for j = 0 to m-1 do  
182     _____rborder.(j) <- 1;
```

```

183 _____ d.(j) <- m;
184 _____done;
185 _____let j= ref 1 and i= ref (m-1) in
186 _____while i.contents >= j.contents do
187 _____ while i.contents >= j.contents && pat.[m-j.contents]=
188 _____ pat.[i.contents-j.contents] do
189 _____j:=j.contents+1;
190 _____rborder.(i.contents-j.contents+1) <- j.contents;
191 _____ done;
192 _____ if j.contents> 1 then (d.(m-j.contents+1) <-
193 _____ min (m- i.contents) d.(m-j.contents+1));
194 _____ i:=i.contents-j.contents+rborder.(m-j.contents+1);
195 _____ j:=max 1 rborder.(m-j.contents+1);
196 _____done;
197 (* computation of the shift-table - phase 2 *)
198 _____let t= ref rborder.(0) and l= ref 1 and s= ref 0 in
199 _____while t.contents > 0 do
200 _____ s:=m-t.contents+1;
201 _____for j=l.contents to s.contents do
202 _____ d.(j) <- min d.(j) s.contents;
203 _____done;
204 _____ t:= rborder.(s.contents);
205 _____ l:=s.contents+1;
206 _____done;
207 _____for i=0 to m-1 do
208 _____ last.(int_of_char pat.[i]) <- i+1;
209 _____done;
210 (* Algorithm of Boyer/Moore *)
211 _____i:=index; j:=m;
212 _____while i.contents<=n-m+1 && j.contents>0 do
213 _____ j:=m;
214 _____ while j.contents >=1 && pat.[j.contents-1]=
215 _____ text.[i.contents+j.contents -2] do
216 _____j:=j.contents-1;

```

```

217 _____ done;
218 _____ if j.contents > 0 then
219 _____ i := i.contents + max d.(j.contents)
220 _____ (j.contents - last.(int_of_char
221 _____ text.[i.contents + j.contents - 2])) ;
222 _____ done;
223 _____ if j.contents > 0 then 0 else i.contents;;

225 (* search string backward using Boyer/More algoritm *)
226 let search_back pat text index =
227 (* computation of the shift-table - phase 1 *)
228 _____ let m = String.length pat and
229 _____ n = String.length text in
230 _____ let rborder = Array.create (m+1) 0 and
231 _____ d = Array.create (m+1) 1 and
232 _____ last = Array.create 256 0 in
233 _____ for j = 0 to m-1 do
234 _____ rborder.(j) <- 1;
235 _____ d.(j) <- m;
236 _____ done;
237 _____ let j = ref 1 and i = ref (m-1) in
238 _____ while i.contents >= j.contents do
239 _____ while i.contents >= j.contents && gch pat (m-j.contents) =
240 _____ gch pat (i.contents - j.contents) do
241 _____ j := j.contents + 1;
242 _____ rborder.(i.contents - j.contents + 1) <- j.contents;
243 _____ done;
244 _____ if j.contents > 1 then (d.(m-j.contents+1) <-
245 _____ min (m - i.contents) d.(m-j.contents+1));
246 _____ i := i.contents - j.contents + rborder.(m-j.contents+1);
247 _____ j := max 1 rborder.(m-j.contents+1);
248 _____ done;
249 (* computation of the shift-table - phase 2 *)
250 _____ let t = ref rborder.(0) and l = ref 1 and s = ref 0 in

```

```

251 _____while t.contents > 0 do
252 _____  s:=m-t.contents+1;
253 _____for j=l.contents to s.contents do
254 _____  d.(j) <- min d.(j) s.contents;
255 _____done;
256 _____  t:= rborder.(s.contents);
257 _____  l:=s.contents+1;
258 _____done;
259 _____for i=0 to m-1 do
260 _____  last.(int_of_char (gch pat i)) <- i+1;
261 _____done;
262 (* Algorithm of Boyer/Moore *)
263 _____i:=n-index; j:=m;
264 _____while i.contents<=n-m+1 && j.contents>0 do
265 _____  j:=m;
266 _____  while j.contents >=1 && (gch pat (j.contents-1)) =
267 _____    (gch text (i.contents+j.contents-2)) do
268 _____    j:=j.contents-1;
269 _____  done;
270 _____  if j.contents> 0 then
271 _____    i:= i.contents+max d.(j.contents)
272 _____    (j.contents-last.(int_of_char (gch text
273 _____    (i.contents+j.contents-2)))) ;
274 _____done;
275 _____if j.contents>0 then 0 else n-i.contents;;

277 (* Graphics routines *)
278 (* Open a graphic window *)
279 let open_gr nm =
280   Graphics.open_graph "" ;
281   Graphics.set_window_title nm;;
282 (* Close a graphic window *)
283 let close_gr ()=
284   Graphics.close_graph();;

```

```

285 (* draw a char (c) on the current graphics screen
286     on the (x,y) position *)
287 let draw_char x y c= Graphics.moveto x y; Graphics.draw_char c;;
288 (* draw a string (s) on the current graphics screen
289     on the (x,y) position *)
290 let draw_string x y s= Graphics.moveto x y; Graphics.draw_string s;;

292 (* Resize the size of the char to extend the line *)
293 let ____line_space=0 and
294     _____caractere_space=1;;
295 let resize_char x = (caractere_space+fst x, line_space+snd x);;

297 let clear_graph ()=
298     let rec loop s= match s with
299         [] -> ()
300         | v::vs -> v.st {cpos= (0,0);
301             tx= ""; stx=0; sty=0; mark=0; mask= ""}
302             (CLR Graphics.red) true; v.draw_self();
303             _____ loop vs
304     in
305     (
306         Graphics.auto_synchronize false;
307         Graphics.clear_graph();
308         Graphics.set_color Graphics.black;
309         loop glb.refresh_list;
310         Graphics.auto_synchronize true);;

311 let tag id rc=
312     _____let (wt, ht)= (120, 25) in
313     _____let (xc, yc)= (rc.x + id*wt, rc.y + rc.h) in
314     _____{x= xc; y= yc; w= wt-2; h= ht};;

316 (* draw a box button on bfc position and bfc size and bw border
317     size and colour t *)

```

```
318 let draw_box t bw bcf =
319   let x1 = bcf.x and y1= bcf.y in
320   let x2= x1+bcf.w and y2= y1+bcf.h in
321   let ix1= x1 + bw and ix2 = x2 - bw and
322     iy1= y1 + bw and iy2= y2 - bw in
323   let border1 g=
324     Graphics.set_color g;
325     Graphics.fill_poly
326       [| (x1, y1); (ix1, iy1); (ix2, iy1);
327         (ix2, iy2);
328         (x2, y2); (x2, y1) |] in
329   let border2 g=
330     Graphics.set_color g;
331     Graphics.fill_poly
332       [| (x1, y1); (ix1, iy1);
333         (ix1, iy2); (ix2, iy2);
334         (x2, y2); (x1, y2)|] in
335   (match t= Graphics.red with
336     true -> Graphics.set_color gray2;
337       Graphics.fill_rect ix1 iy1 (ix2-ix1) (iy2-iy1);
338       border1 gray1;
339       border2 gray3
340     | false -> Graphics.set_color gray2;
341       Graphics.fill_rect ix1 iy1 (ix2-ix1) (iy2-iy1);
342       border1 gray3;
343       border2 gray1);
344   Graphics.set_color Graphics.black;
345   Graphics.draw_rect bcf.x bcf.y bcf.w bcf.h;;

346 (* Draw the text in a window *)
347 let drawText pos_x pos_y c=
348 (* pos_x-> x position of the window, pos_y ->
349   y position of the window, c -> the editor *)
```

```

350   let (char_w, char_h)= resize_char (Graphics.text_size "w")
      in
351   let rec loop i j pos=
352   (* i->line on the screen, j-> column on the screen, pos->
353      position in the string *)
354   if (snd c.cpos)=pos then
355     ( if j< glb.chars_per_line then (draw_char
356   ____ (pos_x+j*char_w-char_w/2)
357         (pos_y-i*char_h) '|' ; c.stx <- j; c.cpos <- ( i,snd c.cpos );)
358     else if j= glb.chars_per_line then (draw_char
359         (pos_x+j*char_w-char_w/2) (pos_y-i*char_h) '|' ;
360         c.stx <- j); c.cpos <- ( i,snd c.cpos ));
361   (* draw the cursor *)
362   match pos with
363   ____ position when (position >= String.length c.tx ) ||
364   ____ (i >= glb.lines_per_page) -> ()
365   (* if end of the string or line position greater than the window,
366      than stop *)
367   ____| position when (c.tx.[position]= '\013') ->
368   ____ draw_char (pos_x+j*char_w) (pos_y-i* char_h) c.tx.[pos];
369   ____ loop (i+1) 0 (position+1)
370   (* if find a <ENTER> caractere than next line *)
371   ____| position when (j > glb.chars_per_line) ->
372   ____loop (i+1) 0 (position)
373   (* if column position greater than window than next line *)
374   ____| _ -> ( if c.mark= pos then ( Graphics.set_color Graphics.red;
375   ____
376   draw_char (pos_x+j*char_w)
377   ____
378   (pos_y-i* char_h) c.tx.[pos];
379   ____
380   Graphics.set_color Graphics.black)
381   ____ else (draw_char (pos_x+j*char_w) (pos_y-i* char_h)
382   ____ c.tx.[pos]));

```

```

380 _____ loop i (j+1) (pos+1)
381 (* or just print the caractere *)
382 in
383     loop 0 0 c.sty;;

385 (* Draw the text in a window *)
386 let drawMask pos_x pos_y c=
387 (* pos_x-> x position of the window, pos_y ->
388     y position of the window, c -> the editor *)
389     let (char_w, char_h)= resize_char (Graphics.text_size "w")
390     in
391     let rec loop i j pos=
392 (* i->line on the screen, j-> column on the screen,
393     pos-> position in the string *)
394     let draw_cursor ()= if (snd c.cpos)=pos then
395         ( if j< glb.chars_per_line then
396             (draw_char (pos_x+j*char_w-char_w/2) (pos_y-i*char_h) '|' );
397             c.stx <- j; c.cpos <- ( i,snd c.cpos );)
398         else if j= glb.chars_per_line then
399             (draw_char (pos_x+j*char_w-char_w/2)
400                 (pos_y-i*char_h) '|' ); c.stx <- j);
401         c.cpos <- ( i,snd c.cpos ););
402     in
403     (* draw the cursor *)
404     match pos with
405     _____ position when (position >= String.length c.tx ) ||
406     _____ (i >= glb.lines_per_page) -> ()
407     (* if end of the string or line position greater
408     than the window, than stop *)
409     _____| position when (c.tx.[position]= '\013') ->
410     _____ draw_char (pos_x+j*char_w) (pos_y-i* char_h)
411     _____ c.tx.[pos];draw_cursor (); loop (i+1) 0 (position+1)
412     (* if find a <ENTER> caractere than next line *)
413     _____| position when (j > glb.chars_per_line) ->

```

```

413 _____draw_cursor (); loop (i+1) 0 (position)
414 (* if column position greater than window than next line *)
415 _____| position when c.mask.[pos]='#' ->
416 _____( if c.tx.[pos]='#' then (Graphics.set_color
417 _____ Graphics.white;
418 _____ Graphics.fill_rect (pos_x+j*char_w)
419 _____ (pos_y-i* char_h) char_w (char_h-1) ;
420 _____ Graphics.set_color Graphics.black;)
421 _____ else (Graphics.set_color Graphics.white;
422 _____ Graphics.draw_rect (pos_x+j*char_w)
423 _____ (pos_y-i* char_h) char_w (char_h-1) ;
424 _____ Graphics.set_color Graphics.black;
425 _____ draw_char (pos_x+j*char_w)
426 _____ (pos_y-i* char_h) c.tx.[pos]););
427 _____draw_cursor (); loop i (j+1) (pos+1); )
428 _____| _ -> draw_char (pos_x+j*char_w) (pos_y-i* char_h)
429 _____c.tx.[pos]; draw_cursor (); loop i (j+1) (pos+1)

431 (* or just print the caractere *)
432 in
433 loop 0 0 c.sty;;

435 (* Mouse routines *)
436 (* Mouse cursor in the box *)
437 let mouse_in_window (x, y) window =
438 if ( (x >= window.x) &&
439 (x < (window.x + window.w)) && (y >= window.y)
440 && (y < (window.y + window.h)))
441 then true else false;;
442 (* Mouse cursor in the window ? *)
443 let click_in_window window =
444 if Graphics.button_down() && mouse_in_window
445 (Graphics.mouse_pos()) window
446 then true

```

```
447     else false;;

449 (* button routines *)
450 (* Get the colour of the button *)
451 let get_color gcolor= match gcolor with
452     (CLR c) -> c
453     | any -> Graphics.red;;
454 (* Get the image of the button *)
455 let get_img image_default image = match image with
456     (IMG i) -> i
457     | any -> image_default;;

459 (* Create a string with n lines of blank text *)
460 let nlines k=
461     let ____ (cw, ch)= resize_char (Graphics.text_size "w") and
462     ____ (size_x, size_y)= (Graphics.size_x(), Graphics.size_y()) in
463     let ____line_len = (size_x - 100) / cw in
464     _____.String.make ((k+1)*line_len) ' ' ;;

466 (* get the index of the Editor *)
467 let getId editor_name =
468     let rec getId_loop id editor_name= match id with
469     ____ i when i > !top -> 0
470     ____ | i when pilha.(i)= editor_name -> i
471     ____ | i -> getId_loop (i+1) editor_name
472     in getId_loop 0 editor_name;;

474 (* load a file into the id Editor *)
475 let load_file id nm=
476     let acc= ref [] in
477     let chan= open_in nm in
478     ( try
479         while true do
480             acc := (input_line chan)::acc.contents
```

```
481         done
482     with
483         any -> close_in chan);
484     (List.rev acc.contents);;

486 (* get the text content of the nm Editor *)
487 let get_contents editor_name= oldStuff.(getId editor_name);;

489 (* set the text content of the nm Editor with s *)
490 let set_contents s editor_name=
491     let id= getId editor_name in
492     oldStuff.(id) <- s;
493     eds.(id).st {cpos=(0,0); tx=oldStuff.(id); stx =0 ; sty=0;
494                 mark=0; mask="" } (CLR Graphics.red) true;
495     eds.(id).draw_self();;

497 let set_mask s editor_name=
498     let id= getId editor_name in
499     oldStuff.(id) <- s;
500     mskStuff.(id) <- (String.copy s);
501     eds.(id).st {cpos=(0,0); tx=oldStuff.(id); stx =0 ; sty=0;
502                 mark=0; mask= (String.copy s) }
503                 (CLR Graphics.red) true;
504     eds.(id).draw_self();;

507 (* Change the name of the Editor nm to newnm *)
508 let change_name nm newnm=
509     let id= getId nm in
510     pilha.(id) <- newnm;
511     let xs= if Sys.file_exists newnm then String.concat ""
512           (load_file id newnm)
513           else String.make 5000 ' ' in
514     ( (*num_of_lines.(id) := (String.length xs ) / 60;*)
```

```
515         set_contents xs newnm);
516     eds.(id).draw_self();;

518 (* Returns the stack of the id(th) element on the stack *)
519 let push label= pilha.(!top) <- label;
520     let i= !top in
521     ( (if !top < (Array.length pilha) - 1 then incr top);
      i);;

525 (* Read just one line: Emacs data input *)

527 type line= {   line_input: (int -> int -> txt  -> unit);
528                 state: (char list -> unit);
529                 draw_line: (unit -> unit)};;
530 type  t =
531     { mutable before : char list;
532       mutable after  : char list };;
533 let xxx= ref "";;

535 let make_line_input  sz rc=
536 let (tw, th)= resize_char (Graphics.text_size "w")  in
537 let charWidth = ref 8 in
538 let stt = ref 0 and maxCharNum = ref sz in
539 let
540     c= {before = []; after = ['-']} and
541     ch= ref ' ' and
542     sc= ref [] and
543     ascii = ref 0 and
544     field_length= Graphics.size_x () - 50 in
545 let rec drawList pos j i s=
546     match s with
547         [] -> ()
```

```

548   | x::xs when j < !stt -> drawList pos (j+1) i xs
549   | x::xs when i <= !maxCharNum ->
550       Graphics.moveto (i * !charWidth) pos;
551       Graphics.draw_char x;
552       drawList pos j (i+1) xs
553   | x::xs -> Graphics.moveto (i * !charWidth) pos;
554       Graphics.set_color Graphics.red;
555       Graphics.draw_char x;
556       Graphics.set_color Graphics.black in
557 let draw_line () =
558   _____Graphics.set_color Graphics.white;
559   _____Graphics.fill_rect rc.x rc.y rc.w rc.h;
560   _____Graphics.set_color Graphics.black;
561   _____Graphics.draw_rect rc.x rc.y rc.w rc.h;
562   _____Graphics.moveto (25+tw) (rc.y+10);
563   _____drawList (rc.y+th/2) 0 4 sc.contents in

565 let backstep c =
566   match c.before with
567       [] -> ()
568   | x :: l -> (
569       c.after <- x::c.after;
570       c.before <- l;
571       if !stt > 0 then stt := !stt - 1
572   ) in
573 let step c =
574   match c.after with
575       [] -> ()
576   | x :: l ->
577       (
578       c.before <- x::c.before;
579       c.after <- l;
580       if List.length c.before > !maxCharNum then
581       stt := !stt + 1

```

```
582         ) in
583 let delete c=
584   match c.before with
585     [] -> ()
586   | x::xs -> c.before <- xs;
587               if !stt > 0
588               then stt := !stt - 1 in
589 let insert c x =
590   if (List.length c.before) >= !maxCharNum then
591     stt := !stt + 1;
592     c.before <- x::c.before in
593 let get_all c = (List.rev c.before) @ (' ':: c.after) in
594 let get_chars c = (List.rev c.before) @ c.after in

596 let get_string c=
597   let
598     i = ref 0 and
599     s= ref (get_chars c) in
600   let
601     str= String.create (List.length !s) in
602   begin
603     while !s <> [] do
604       str.[!i] <- List.hd (!s);
605       s := List.tl s.contents;
606       incr i
607     done;
608     str
609   end in
610 let savt nm s =
611   let chan= open_out nm in
612     output_string chan s;
613     close_out chan
614 in let stpYes= ref true
615 and enter= ref 0
```

```

616 and erase_sc= ref false in
617 let read_string s_or_x id xs=
618   maxCharNum := field_length / !charWidth - 1;
619   ch := Graphics.read_key();
620   c.before <- []; c.after <- [];
621   stpYes := false;
622   if (s_or_x= 1) then enter := 3
623   else (if s_or_x= 2 then enter := 4 else enter := 0);
624   erase_sc := false;
625   while enter.contents <> 5 do
626     (*ch := Graphics.read_key(); *)
627     if (int_of_char !ch) < 32 then
628       ( );
629     ( match (int_of_char !ch) with
630       2 -> backstep c
631       | 14 -> step c (* 6 is Needed for something else; anyway,
632                     I can live without it *)
633       | 8 -> delete c
634   _____ | 13 when enter.contents= 0 ->
635                 c.before <- [];
636                 c.after <- ['Q'; 'u'; 'i'; 't'];
637                 enter := 5
638       | 13 when enter.contents = 3 ->
639                 let rg= get_string c and
640                 (i, j)= xs.cpos in
641                 ( ( try
642                     let pos=search rg xs.tx (j+2) in
643                     (* using the STR library on Ocaml ->
644                       Str.search_forward (Str.regexp rg)*)
645                     glb.search_pos <- pos
646                     with
647                     any -> glb.search_pos <- j
648                   )
649                 );

```

```

650             c.before <- [];
651     c.after <- ['D'; 'o'; 'n'; 'e'];
652             enter := 5
653     | 13 when enter.contents = 4 ->
654     _____
        let rg= get_string c and
655     _____
        (i, j)= xs.cpos in
656     _____                ( ( try
657     _____
        let pos= search_back rg xs.tx j in
658     _____
        (* using the STR library on Ocaml ->
659     _____
        Str.search_backward (Str.regexp rg)*)
660     _____
        glb.search_pos <- pos
661     _____                with
662     _____
        any -> glb.search_pos <- j
663     _____                )
664     _____                );
665     _____                c.before <- [];
666     c.after <- ['D'; 'o'; 'n'; 'e'];
667             enter := 5
668     | 13 when enter.contents= 1 && id >= 0 ->
669             let newnm= get_string c in
670             change_name pilha.(id) newnm;
671     _____c.before <- [];
672     _____c.after <- ['L'; 'o'; 'a'; 'd'; 'e'; 'd'];
673     _____enter := 5
674     | 13 when enter.contents= 2 && id >= 0 ->
675     _____
        let newnm= get_string c in

```

```

676 _____ pilha.(id) <- newnm;
677 _____
    savt (pilha.(id)) xs.tx;
678 _____ c.before <- [];
679 _____
    c.after <- ['W'; 'r'; 'i'; 't'; 't'; 'e'; 'n'];
680 _____ enter := 5
681 _____ | 19 when id >= 0 -> savt (pilha.(id)) xs.tx;
682 _____ c.before <- [];
683 _____ c.after <- ['S'; 'a'; 'v'; 'e'; 'd'];
684 _____ enter := 5
685 _____ | 23 -> stpYes := true;
686 _____ enter := 2;
687 _____ c.before <- [];
688 _____ c.after <- ['C'; 'n'; 't'; 'r'; 'l'; '-' ; 'W'];
689 _____ erase_sc := true
690 _____ | 6 when stpYes.contents -> step c
691 _____ | 6 -> stpYes := true;
692 _____ enter := 1;
693 _____ c.before <- [];
694 _____ c.after <- ['C'; 'n'; 't'; 'r'; 'l'; '-' ; 'F'];
695 _____ erase_sc := true
696 _____ | 27 -> ()

698 _____ | 1 -> print_int !ascii;
699 _____ print_newline()
700 _____ | i when i > 31 -> insert c !ch
701 _____ | - -> ());
702 _____ ascii := int_of_char !ch;
703 _____ sc := get_all c;
704 _____ draw_line ();
705 _____ if !enter <> 5 then (ch := Graphics.read_key());
706 _____ if erase_sc.contents then (c.before <- [];
707 _____ c.after <- []; erase_sc := false);

```

```
710  done;
711  ch := ' -';
712  xxx := get_string c;
713  (*c.before <- []; c.after <- [] *) in
714  {line_input=read_string; state= (fun cs -> (sc := cs));
715    draw_line= draw_line };;

718  (*eMac like Text editor *)

720  (*Text editor *)
721  let make_edt
722  _____?(fn=fun s -> s)
723  _____?(the_text= "")
724  _____name (x0, y0, w, h)=
725  let sts= {_____bkpx=0;
726    _____nm= name;
727    (* toggle to delete the char or not when using the backspace key *)
728    _____bkerase= true;
729    _____ctx_save= ""; cmk_save="";
730    (* buffer to save the response of CTRL-T *)
731    _____sty_stk= [];
732    _____insertions= false;
733    _____etx= {
734      _____tx= "";
735      _____cpos= (0,0);
736      _____stx=0;
737      _____sty=0; mark=0; mask= "" };
738    _____virtual_char= 0} in
739  let (tw, th)= resize_char (Graphics.text_size "w") and
740  _____rc = mkrec x0 y0 w h in
741  let id= push sts.nm and
```

```

742 (* stack control of the sty - scrool routines *)
743 _____push_y v=
744   _____ if sts.sty_stk <> [] && v= List.hd sts.sty_stk then ()
745   _____else (sts.sty_stk <- v::sts.sty_stk)
746 and pop_y () = match sts.sty_stk with
747   _____ [] -> 0
748   _____| (x::xs) -> sts.sty_stk <- xs; x
749 in

751 (* Cursor movements *)

753 (* find the end of the text *)
754 let last_position c =
755   let rec find_last_char_non_space txt position =
756     match txt.[position] with
757       ch when ch=' ' -> find_last_char_non_space txt (position-1)
758       | _ -> (position)
759   in find_last_char_non_space c.tx (String.length c.tx -1)
760 in

761 (* reset the screen editor *)
762 let reset_editor c=
763   _____c.cpos <- (0,0); sts.sty_stk <- []; c.stx <- 0; c.sty <- 0
764 in

765 (* swap ctx-buffer and c.tx routine *)
766 let ctx_swap c=
767   if sts.ctx_save= "" then ()
768   else ( let ctx= sts.ctx_save in
769     _____sts.ctx_save <- c.tx;
770     _____c.tx <- ctx;
771     _____reset_editor c)
772 in

773 (* find the start of the next line *)
774 let rec next_line position texto chars_per_line =
775   match String.length texto with

```

```

776 _____ j when position>=j -> position;
777 _____| - when chars_per_line <=0 -> position+1;
778 _____| - when texto.[position] = '\013' -> position+1;
779 _____| - -> next_line (position+1) texto (chars_per_line - 1)
780 in
781 (* Scroll the screen *)
782 let scroll c =
783 _____push_y c.sty;
784 _____c.sty <- (next_line c.sty c.tx glb.chars_per_line);
785 in
786 let scroll_back c =
787 _____c.sty <- pop_y ();
788 in
789 (* go a step forward *)
790 let stp c = match c.cpos with
791 _____ (i,j) when j >= (String.length c.tx) -> ()
792 (* if out or end of the string do nothing *)
793 _____| (i,j) when c.tx.[j] = '\013' &&
794 _____ i < (glb.lines_per_page- 1) -> c.stx <- 0;
795 _____ c.cpos <- (i+1,j+1);
796 (* if found newline and its not end of the line just go to
797 next line *)
798 _____| (i,j) when c.tx.[j] = '\013' &&
799 _____ i >= (glb.lines_per_page- 1) ->
800 scroll c ; c.stx <- 0; c.cpos <- (i,j+1);
801 (* if found newline and its end of the window do a scroll *)
802 _____| (i,j) when c.stx >= glb.chars_per_line
803 _____ && i < (glb.lines_per_page- 1) ->
804 c.stx <- 0; c.cpos <- (i+1,j+1);
805 (* if end of line (window) and do not end of screen go
806 next line *)
807 _____| (i,j) when c.stx >= glb.chars_per_line
808 _____ && i >= (glb.lines_per_page- 1) ->
809 scroll c; c.stx <- 0; c.cpos <- (i,j+1);

```

```
810 (* if end of line (window) and end of screen do scroll *)
811 _____| (i,j) -> c.cpos <- (i , j+ 1); c.stx <- c.stx+1;
812 (* just go next step *)
813 in
814 (* go a step backward *)
815 let bkstep c =
816 _____match c.cpos with
817 _____ (i,j) when j<=0 -> ()
818 (* if start of the string do nothing *)
819 _____| (i,j) when c.tx.[j-1] = '\013' && i<=0 ->
820     scroll_back c; c.cpos <- (i,j-1);
821 (* if find newline and first line scroll back *)
822 _____| (i,j) when c.tx.[j-1] = '\013' && i>0 ->
823     c.cpos <- (i-1,j-1);
824 (* if find newline just go back *)
825 _____| (i,j) when c.stx = 0 && i<=0 ->
826     scroll_back c; c.cpos <- (i,j-1);
827 (* if first position and first line scroll back *)
828 _____| (i,j) when c.stx <=0 -> c.cpos <- (i-1,j-1);
829 (* if first position and not first line go back *)
830 _____| (i,j) -> c.cpos <- (i,j-1);
831 in
832 (* goto the position on the text *)
833 let rec goto_position c position=
834     match c.cpos with
835         (i, j) when position <0 || j> position -> ()
836         | _ -> stp c; goto_position c position
837 in
838 (* goto the end of the document *)
839 let goto_end_of_doc c= reset_editor c;
840     goto_position c (last_position c)
841 in
842 (* goto n positions back *)
843 let rec goto_pos_bk c pos =
```

```

844 _____if (c.stx <=pos) then ()
845 _____else (bkstep c; c.stx <-
      c.stx -1; goto_pos_bk c pos)
846 in
847 (* goto first position of the line *)
848 let rec goto_first_pos_of_line c =
849 _____c.cpos <- ((fst c.cpos), (snd c.cpos - c.stx)); c.stx <- 0;
850 in
851 (* go previous line step 1 *)
852 let upstp c =
853 _____sts.bkpx <- c.stx;
854 _____c.cpos <- ((fst c.cpos), (snd c.cpos - c.stx)); c.stx <- 0;
855 _____bkstep c;
856 _____sts.virtual_char <- 31;
857 in
858 (* go previous line step 2 *)
859 let upstp_2 c =
860 _____goto_pos_bk c sts.bkpx;
861 in
862 (* goto on a position in the line *)
863 let rec goto_pos c pos = match c.cpos with
864 _____ (i, j) when j>=(String.length c.tx) -2 -> ()
865 _____| (i, j) when c.stx>= glb.chars_per_line &&
866 _____pos>= glb.chars_per_line -> ();
867 _____| (i, j) when c.stx>= glb.chars_per_line -> ()
868 _____| (i, j) when c.stx>=pos -> ()
869 _____| (i, j) when c.tx.[(snd c.cpos)]= '\013' -> ()
870 _____| _ -> stp c; goto_pos c pos
871 in
872 (* goto end of the line *)
873 let goto_end_of_line c=
874 _____( if c.stx = glb.lines_per_page then () );
875 _____goto_pos c ( glb.chars_per_line );
876 in

```

---

```

877 let rec nxt_line_loop c=
878 _____if c.stx=0 || (snd c.cpos)> (String.length c.tx)- 2 then ()
879 _____else (stp c; nxt_line_loop c)
880 in
881 (* goto first position of the next line *)
882 let nxt_line c = stp c; nxt_line_loop c;
883 in
884 (* go down one line on the same position *)
885 let dwnstp c =
886     sts.bkpx <- c.stx; nxt_line c; goto_pos c sts.bkpx;
887 in
888 (* insert a char on the text *)
889 let rec ins_char ch c = match c.cpos with
890 _____ (i,j) when j> (String.length c.tx) -2 -> ()
891 _____| (i,j) ->
892         let x= c.tx in
893         let
894             tr= (String.length x) - 2 - j and
895             fr= (j+1) in
896             ( String.blit x j x fr tr;
897               x.[j] <- ch; stp c)

899 in
900 let ins_txt c txt=
901     for i= 0 to String.length txt - 1 do
902         ins_char txt.[i] c
903     done
904 in
905 let del_char c = match c.cpos with
906 _____(i,j) -> String.blit c.tx (j+1) c.tx j ((String.length c.tx)
    - j - 1);
907 _____c.tx.[String.length c.tx -1] <- ' '
908 in
909 (* delete j caracteres from the cursor *)

```

```
910 let rec del_them j c=
911     if j < 1 then ()
912     else ( del_char c; del_them (j-1) c )
913 in
914 (* delete n caracteres back the cursor *)
915 let rec del_back n c=
916     if n < 0 then ()
917     else (bkstep c; del_char c; del_back (n-1) c)
918 in
919 (* delete from mark to cursor *)
920 let del_txt c = match c.cpos with
921   _____(i, j) when j > c.mark ->
922     ins_char (char_of_int 13) c ;del_back (j - c.mark) c
923   _____ | (i, j) -> del_them (c.mark - j) c
924 in
925 (* Reset the cursor on the display *)
926 let reset_display c = c.cpos <- (0, c.sty); c.stx <- 0;
927 in
928 (* put the cursor on the mouse position *)
929 let mouse_point c =
930     let rec loop i =
931   _____if i<=0 then ()
932   _____else (dwnstp c; loop (i-1) ) in
933     let (x,y)= glb.mouse_position in
934     let (cx, cy)= (min glb.chars_per_line ((x-rc.x-3+caractere_space)/tw)
935       , min glb.lines_per_page ((rc.y+rc.h-3+th-y)/th)) in
936     reset_display c;
937     loop (cy-1);
938     goto_first_pos_of_line c ; goto_pos c cx;
939 (* put the cursor at the cx position *)
940 in
941 (* find the position of the char after the cursor *)
942 let find_char char c=
943     let rec loop j=
```

```

944      if (j>String.length c.tx -1) || (c.tx.[j]=char) then j
945      else loop (j+1) in
946    loop (snd c.cpos)
947  in
948  let rec find_to j n str= match n with
949      t when t < (String.length str) - 2 &&
950      str.[t] = '\009' && str.[t+1]= ' ' ->
951          t - j
952      | t when t < (String.length str) - 2 -> find_to j (n+1) str
953      | t -> (String.length str) - 2 - j
954  in
955  let contour= ref (Graphics.red)
956  (*and edTxt= ref {tx= ""; cpos= (0,0); stx=0; sty=0} *) in

958  let set_state cpos gcor i=
959      sts.etx <- cpos;
960      contour := (get_color gcor);
961      if i then ( sts.etx.sty <- 0;
962      sts.etx.stx <- 0; sts.sty_stk <- [] ) in
963  let (size_x , size_y)= (Graphics.size_x(), Graphics.size_y()) in
964  let ctrl_x= make_line_input 20
965      (mkrec 25 (rc.y - 2*th) (size_x - 50) (2*th))
966  in
967  let process_esc k c= match c.cpos with
968      _____ (i , j) when k= ' ' -> c.mark <- j
969      _____| (i , j) when k= 'w' && j > c.mark ->
970      _____          push_circular ( String.sub c.tx
          c.mark (j- c.mark))
971      _____| (i , j) when k= 'w' && c.mark > j ->
972      _____          push_circular ( String.sub c.tx
          j (c.mark - j))
973      _____| (i , j) when k= 'a' -> reset_editor c
974      _____| (i , j) when k= 'e' -> goto_end_of_doc c
975      _____| (i , j) when k= 'y' && glb.mrk>=0 && glb.mrk < j ->

```

```
976 _____ c.mark <- glb.mrk; del_txt c; circula();
977 _____ ins_txt c (getop ())
978 _____| (i, j) -> ()
979 (* the position of the tag box *)
980 and tg= tag id rc
981 (* Control a virtual keypress *)
982 and pop_virtualchar () =
983     let c= sts.virtual_char in ( sts.virtual_char <- 0; c )
984 in
985 (* Control of mini buttons in the window editor *)
986 (* position of the mini buttons *)
987 let mini_rc = {x= rc.x+rc.w-15; y= rc.y+rc.h-15; w= 15; h= 15 }
988 and scroll_up_bt = {x= rc.x+rc.w-15; y= rc.y+rc.h-40; w= 15; h= 15 }
989 and scroll_down_bt = {x= rc.x+rc.w-15; y= rc.y; w= 15; h= 15 }
990 in
991 (* draw of tre mini button *)
992 let draw_mini_buttons () =
993     Graphics.draw_rect mini_rc.x
994     mini_rc.y mini_rc.w mini_rc.h;
995     Graphics.draw_rect scroll_up_bt.x
996     scroll_up_bt.y scroll_up_bt.w scroll_up_bt.h;
997     Graphics.draw_rect scroll_down_bt.x
998     scroll_down_bt.y scroll_down_bt.w scroll_down_bt.h;
999     Graphics.moveto (mini_rc.x + 3)
1000     mini_rc.y; Graphics.draw_char 'X' ;
1001     Graphics.moveto (scroll_up_bt.x+1)
1002     (scroll_up_bt.y+1);
1003     Graphics.lineto (scroll_up_bt.x+scroll_up_bt.w/2)
1004     (scroll_up_bt.y+scroll_up_bt.h/2+4);
1005     Graphics.lineto (scroll_up_bt.x+scroll_up_bt.w-1)
1006     (scroll_up_bt.y);
1007     Graphics.moveto (scroll_down_bt.x)
1008     (scroll_down_bt.y+scroll_down_bt.h);
1009     Graphics.lineto (scroll_down_bt.x+scroll_down_bt.w/2+1)
```

```
1010      (scroll_down_bt.y+scroll_down_bt.h/2-4);
1011      Graphics.lineto (scroll_down_bt.x+scroll_down_bt.w)
1012      (scroll_down_bt.y+scroll_down_bt.h);in
1013  let
1014      draw_editor ()=
1015          Graphics.auto_synchronize false;
1016          let tg= tag id rc in
1017              draw_box contour.contents 2 tg;
1018              Graphics.set_color background_edt;
1019              Graphics.fill_rect rc.x rc.y rc.w rc.h;
1020              Graphics.set_color contour.contents;
1021              Graphics.draw_rect rc.x rc.y rc.w rc.h;
1022              draw_mini_buttons ();
1023              Graphics.set_color Graphics.black;
1024              draw_string (tg.x+6) (tg.y+ (tg.h-th)/2) (pilha.(id));
1025              drawText (rc.x+3) (rc.y + rc.h - th- 3) sts.etx;
1026              ctrl_x.draw_line ();
1027              Graphics.auto_synchronize true
1028  and
1029      in_muride ()=
1030          if mouse_in_window (Graphics.mouse_pos()) tg then
1031              (sts.insertions <- true; char_of_int 21)
1032          else (sts.insertions <- false; char_of_int 0)
1033  and
1034      in_char ()=
1035          if sts.virtual_char <> 0 then char_of_int (pop_virtualchar () )
1036          (* if there are any key on the virtual keybord then return
1037          the top one *)
1038          else begin
1039              let ev= Graphics.wait_next_event [Graphics.Button_down;
1040              Graphics.Key_pressed] in
1041              (* wait for a key pressed or a mouse button down *)
1042              glb.mouse_position <- Graphics.mouse_pos();
1043              (* storage the position of the mouse *)
```

```

1044 _____match ev with
1045 _____   event when event.Graphics.keypressed -> ev.Graphics.key
1046 (* if a key is pressed then return this key *)
1047 _____| event when (mouse_in_window glb.mouse_position tg) ->
1048 _____   sts.insertions <- true; char_of_int 0
1049 (* if click on the tag active the editor id *)
1050 _____| event when (mouse_in_window glb.mouse_position scroll_up_bt)
1051 _____-> sts.insertions <- true; char_of_int 21
1052 (* if click on scroll down button do it *)
1053 _____| event when (mouse_in_window
1054 _____glb.mouse_position scroll_down_bt) ->
1055 _____   sts.insertions <- true; char_of_int 09
1056 (* if click on scroll up button do it *)
1057 _____| event when (mouse_in_window glb.mouse_position
1058 _____   mini_rc) ->
1059 _____   char_of_int 20
1059 (* if click on the mini-button then return a ascii 20 *)
1060 _____| event when (mouse_in_window glb.mouse_position rc) ->
1061 _____   sts.insertions <- true; char_of_int 28
1062 (* if mouse in the editor window then move the cursor
1063 to it position on keypressed ascii 28 *)
1064 _____| _ -> sts.insertions <- false; char_of_int 0
1065 end
1066 in
1067   set_state {tx= "";   cpos= (0,0); stx=0; sty=0; mark=0;
1068             mask= ""} (CLR Graphics.red) true;
1069   ctrl_x.state [' -'];
1070   draw_editor ();

1072 (* This is the process event! *)
1073 let text_input ()=
1074 let lines = if oldStuff.(id) = "" then the_text
1075             else oldStuff.(id) in
1076 let

```

```

1077   c=createText (if lines="" then (String.make 500 ' ') else lines)
1078   and ch= ref (char_of_int 0) in
1079   (
1080     c.stx <- 0;
1081     c.sty <- 0;
1082     ch :=in_muride ();
1083     while sts.insertions do
1084       if !ch<> char_of_int 21
1085       then ch := in_char () else (ch:= char_of_int 0);
1086       if !ch <> (char_of_int 27) then glb.mrk <- (-1) else ();
1087       (*mainkeys*)
1088       ( match (int_of_char !ch) with
1089 _____ 01 (*Ctrl-a Goto first position of the line *) ->
1090 _____ goto_first_pos_of_line c
1091 _____| 02 (*Ctrl-b - Backstep *) -> bkstep c
1092 _____| 03 (*Ctrl-c *) -> ()
1093 _____| 04 (*Ctrl-d Backspace *) -> del_char c
1094 _____| 05 (*Ctrl-e goto end of line*) -> goto_end_of_line c
1095 _____| 06 (*Ctrl-f *)-> stp c
1096 _____| 07 (*Ctrl-g *) -> ()
1097 _____| 08 (*Ctrl-h & Backspace *) -> if (snd c.cpos)>0 then
1098 _____( bkstep c ; if sts.bkerase then del_char c)
1099 _____| 09 (*Ctrl-i *) -> scroll c;sts.virtual_char <- 14;
1100 _____| 10 (*Ctrl-j insert a line break *) ->
1101 _____ins_char (char_of_int 13) c (* standard behavior *)
1102     (* nxt_line c *)
1103     (* non standard behavior *)
1104 _____| 11 (*Ctrl-k kills to the end of line *) ->
1105 _____c.mark <- find_char '\013' c;
1106 _____if c.mark= (snd c.cpos) then del_char c
1107 _____else (process_esc 'w' c; del_txt c)
1108 _____| 12 (*Ctrl-m *) -> ()
1109 _____| 13 (*ENTER *) -> ins_char (char_of_int 13) c
1110 _____| 14 (*Ctrl-n Move to next line *) -> dwnstp c;

```

```

1112 _____| 15 (*Ctrl-o *) ->ins_char (char_of_int 13) c; bkstep c
1113 _____| 16 (*Ctrl-P Move to previous Line Step-1*) -> upstp
      c
1114 _____| 31 (*Ctrl-P Move to previous Line Step-2*) -> upstp_2 c
1115 _____| 17 (*Ctrl-q Toggle between bkstep and bkerase *) ->
1116 _____ sts.bkerase <- not sts.bkerase
1117 _____| 18 (*Ctrl-r search text backward *) ->
1118 _____ctrl_x.line_input 2 id c;
1119 _____reset_editor c; goto_position c (glb.search_pos -2);
1120 _____| 19 (*Ctrl-s search text forward *) ->
1121 _____ctrl_x.line_input 1 id c;
1122 _____reset_editor c; goto_position c (glb.search_pos -2)
1123 _____| 20 (*Ctrl-t: Applica função *) ->
1124 _____sts.ctx_save <- c.tx;
1125 _____reset_editor c;
1126 _____let sz= String.length c.tx and
1127 _____      new_tx= fn c.tx in
1128 _____let new_sz= String.length new_tx in
1129 _____if new_sz > sz then ()
1130 else ( c.tx <- new_tx ^ (String.make (sz- new_sz) ' '))
1131 _____| 21 (*Ctrl-u *) -> scroll_back c; sts.virtual_char <- 16;
1132 _____| 22 (*Ctrl-v *) -> ctx_swap c
1133 _____| 23 (*Ctrl-w Erase marked area *) ->
      process_esc 'w' c; del_txt c
1134 _____| 24 (*Ctrl-x Execute function *) ->
      ctrl_x.state [ 'C'; 'n'; 't'; 'r'; 'l'; '-' ; 'X' ];
1135 _____
      ctrl_x.draw_line ();
1136 _____
      oldStuff.(id) <- c.tx;
1137 _____
      ctrl_x.line_input 0 id c;

```

```

1138 _____
      c.tx <- oldStuff.(id);
1139 _____
      reset_editor c;
1140 _____
      draw_editor()

1142 _____| 25 (*Ctrl-y paste the buffer *) when getop () <> "" ->
1143 _____|          let (i, j)= c.cpos in
1144 _____|          glb.mrk <- j;
1145 _____|          ins_txt c (getop ())
1146 _____| 26 (*Ctrl-z *) -> sts.insertions <- false
1147 _____| 27 (* Esc *) -> process_esc (Graphics.read_key()) c
1148 _____| 28          -> mouse_point c;
1149 _____| x when ch.contents >= ' '
      (* Insert char *) ->
1150 _____ ins_char ch.contents c (**)
1151 _____| - -> ());
1152      set_state c (CLR Graphics.blue) false;
1153      draw_editor();
1154      done;
1155      if !contour= Graphics.blue then
1156      (set_state c (CLR Graphics.red) true; draw_editor())
1157      else ();
1158      sts.insertions <- false;
1159      c.stx <- 0;
1160      c.sty <- 0;
1161      sts.sty_stk <- []
1162  ) in oldStuff.(id) <- the_text;
1163      eds.(id) <- {process_event=text_input;
1164      st= set_state; draw_self= draw_editor};
1165      glb.refresh_list <- eds.(id)::glb.refresh_list;
1166      eds.(id)
1167  ;;

```

```
1169 let mk_edt ?(txt= String.make 400 ' ')
1170   ?(fn= fun s -> s)
1171   nm=
1172   let (cw, ch)= resize_char
1173   (Graphics.text_size "w") in
1174   let (size_x, size_y)=
1175   (Graphics.size_x(), Graphics.size_y()) in
1176   let rc= (25, size_y -
1177   (winHeight+6)*ch, size_x - 50, 100) in
1178   glb.chars_per_line <-
1179   (size_x - 100) / cw;
1180   make_edt ~the_text:txt
1181   ~fn:fn nm rc
1182   ;;

1184 let make_button fn txt (x0, y0)=
1185 let (cw, ch)= Graphics.text_size "w" in
1186 let (w, h)= (12*cw, 2*ch) in
1187 let (tw, th)= Graphics.text_size txt and
1188   rc = mkrec x0 y0 w h and
1189   pressed = ref false and
1190   stat= ref {cpos= (0,0); tx=txt; stx=0; sty=0; mark=0; mask= ""}
1191   and contour= ref Graphics.red in
1192 let draw ()=
1193   Graphics.auto_synchronize false;
1194   draw_box !contour 3 rc;
1195   Graphics.set_color Graphics.black;
1196   draw_string (x0+(w-tw)/2) (y0+(h-th)/2) !stat.tx ;
1197   Graphics.auto_synchronize true
1198 and set_state c gcor i= contour:= (get_color gcor) in
1199   draw ();
1200   let button_input ()=
1201     if (click_in_window rc) &&
```

```
1202     not pressed.contents then (fn(); pressed := true;
1203     set_state !stat (CLR Graphics.blue) false; draw());
1204     if not(Graphics.button_down()) && pressed.contents
1205     then (pressed := false;
1206     set_state !stat (CLR Graphics.red) false; draw ())
1207     in
1208     let theButton= {process_event=button_input;
1209     st= set_state; draw_self= draw}
1210     in ( glb.refresh_list <- theButton::glb.refresh_list;
1211     theButton)
1212     ;;

1214 let refresh ()=
1215     let rec loop s= match s with
1216         [] -> ()
1217         | v::vs -> v.st {cpos= (0,0);
1218         tx= ""; stx=0; sty=0; mark=0; mask= "" }
1219         (CLR Graphics.red) true; v.draw_self(); loop vs
1220     in
1221     ignore(Graphics.wait_next_event [Graphics.Mouse_motion;
1222     Graphics.Button_down]);
1223     let (sx,sy)= (Graphics.size_x(), Graphics.size_y()) in
1224     if last_sz.contents = (0,0) then
1225         ( Graphics.auto_synchronize false;
1226         set_siz();
1227         loop glb.refresh_list;
1228         Graphics.auto_synchronize true )
1229     else ( if (sx,sy) <> last_sz.contents then
1230         ( Graphics.auto_synchronize false;
1231         Graphics.set_color background_mask;
1232         Graphics.fill_rect 0 0 sx sy;
1233         Graphics.set_color Graphics.black;
1234         set_siz();
1235         loop glb.refresh_list;
```

```
1236           Graphics.auto_synchronize true
1237         )
1238     )
1239 ;;

1241 (* default Exit Button *)
1242 let exit_button ()=
1243 let (cw, ch)= Graphics.text_size "w" in
1244 let (w, h)= (12*cw, 2*ch) in
1245 let (size_x, size_y)= (Graphics.size_x(), Graphics.size_y()) in
1246 let rc= (size_x - 25 - w, size_y-(winHeight+8)*ch - h -10) in
1247     make_button (fun () -> Graphics.close_graph(); exit 0) "Exit" rc
1248 ;;

1250 let button i fs nm=
1251 let (cw, ch)= Graphics.text_size "w" in
1252 let (w, h)= (15*cw, 2*ch) in
1253 let (size_x, size_y)= (Graphics.size_x(), Graphics.size_y()) in
1254 let rc= (50+ i* w, size_y-(winHeight+8)*ch - h -10) in
1255     make_button fs nm rc
1256 ;;

1258 let next_button_row i fs nm=
1259 let (cw, ch)= Graphics.text_size "w" in
1260 let (w, h)= (15*cw, 2*ch) in
1261 let (size_x, size_y)= (Graphics.size_x(), Graphics.size_y()) in
1262 let rc= (50+ i* w, size_y-(winHeight+8)*ch - 2*h -10) in
1263     make_button fs nm rc;;

1265 let get_fields msk txt=
1266     match String.length txt with
1267     sz when sz <> (String.length msk) -> txt
1268     | sz -> let c= String.make sz ' ' in
1269         for i= 0 to sz - 1 do
```

```

1270             if msk.[i]= '#' then c.[i] <-
1271             if txt.[i]='#' then ' ' else txt.[i]
1272             else c.[i] <- ' '
1273         done; c;;

1275 (* Mask Editor *)
1276 let make_entry
1277   _____?(fn=fun s -> s)
1278   _____?(pp= get_fields)
1279   _____?(the_text= "")
1280   _____(*?(mask=fun x -> true) *)
1281   _____name (x0, y0, w, h)=
1282   let sts= { _____bkpx=0;
1283   _____nm= name;
1284   (* togle to delete the char or not when using the backspace key *)
1285   _____bkerase= true;
1286   _____ctx_save= "";
1287   _____cmk_save="";
1288   _____sty_stk= [];
1289   _____insertions= false;
1290   _____etx= {__tx= ""};
1291   _____cpos= (0,0);
1292   _____stx=0;
1293   _____sty=0; mark=0; mask="" };
1294   _____virtual_char= 0} and
1295   _____ed_lines= the_text in
1296   let (tw, th)= resize_char (Graphics.text_size "w") and
1297   _____rc = mkrec x0 y0 w h in
1298   let id= push sts.nm and
1299   (* stack control of the sty - scrool routines *)
1300   _____push_y v=
1301   _____ if sts.sty_stk <> [] && v= List.hd sts.sty_stk then ()
1302   _____ else (sts.sty_stk <- v::sts.sty_stk)
1303   and pop_y () = match sts.sty_stk with

```

```

1304 _____ [] -> 0
1305 _____| (x::xs) -> sts.sty_stk <- xs; x
1306 in

1308 (* Cursor movements *)

1310 (* find the end of the text *)
1311 let last_position c =
1312   let rec find_last_char_non_space txt position =
1313     match txt.[position] with
1314     _____ ch when ch=' ' -> find_last_char_non_space txt (position- 1)
1315     _____ | _ -> (position)
1316   in find_last_char_non_space c.tx (String.length c.tx -1)
1317 in
1318 (* reset the screen editor *)
1319 let reset_editor c=
1320   _____c.cpos <- (0,0); sts.sty_stk <- []; c.stx <- 0; c.sty <- 0
1321 in
1322 (* swap ctx-buffer and c.tx routine *)
1323 let ctx_swap c=
1324   if sts.ctx_save= "" then ()
1325   else ( let ctx= sts.ctx_save and
1326     _____ cmk= sts.cmk_save in
1327     _____sts.ctx_save <- c.tx;
1328     _____sts.cmk_save <- c.mask;
1329     _____c.tx <- ctx;
1330     _____c.mask <- cmk;
1331     _____reset_editor c)
1332 in
1333 (* find the start of the next line *)
1334 let rec next_line position texto chars_per_line =
1335   match String.length texto with
1336   _____ j when position>=j -> position;
1337   _____| _ when chars_per_line <=0 -> position+1;

```

```

1338 _____| - when texto.[position] = '\013' -> position+1;
1339 _____| - -> next_line (position+1) texto (chars_per_line - 1)
1340 in
1341 (* Scroll the screen *)
1342 let scroll c =
1343 _____push_y c.sty;
1344 _____c.sty <- (next_line c.sty c.tx glb.chars_per_line);
1345 in
1346 let scroll_back c =
1347 _____c.sty <- pop_y ();
1348 in
1349 (* go a step forward *)
1350 let stp c = match c.cpos with
1351 _____ (i,j) when j >= (String.length c.tx) -> ()
1352 (* if out or end of the string do nothing *)
1353 _____| (i,j) when c.tx.[j] = '\013' &&
1354 _____ i < (glb.lines_per_page- 1) ->
1355 _____ c.stx <- 0; c.cpos <-
1356 _____ (i+1,j+1);
1357 (* if found newline and its not end of the line just go to next line *)
1358 _____| (i,j) when c.tx.[j] = '\013' &&
1359 _____ i >= (glb.lines_per_page- 1) ->
1360 _____ scroll c ; c.stx <- 0; c.cpos
1361 _____ <- (i,j+1);
1362 (* if found newline and its end of the window do a scroll *)
1363 _____| (i,j) when c.stx >= glb.chars_per_line
1364 _____ && i < (glb.lines_per_page- 1) ->
1365 _____ c.stx <- 0; c.cpos <- (i+1,j+1);
1366 (* if end of line (window) and do not end of screen go next line *)
1367 _____| (i,j) when c.stx >= glb.chars_per_line
1368 _____ && i >= (glb.lines_per_page- 1) ->
1369 _____ scroll c ; c.stx <- 0; c.cpos <- (i,j+1);
1370 (* if end of line (window) and end of screen do scroll *)
1371 _____| (i,j) -> c.cpos <- (i , j+ 1); c.stx <- c.stx+1;

```

```
1370 (* just go next step *)
1371 in
1372 (* go a step backward *)
1373 let bkstep c =
1374   _____match c.cpos with
1375   _____ (i,j) when j<=0 -> ()
1376   (* if start of the string do nothing *)
1377   _____| (i,j) when c.tx.[j-1] = '\013' && i<=0 ->
1378   _____c.sty <- pop-y (); c.cpos <- (0,j-1);
1379   (* if find newline and first line scroll back *)
1380   _____| (i,j) when c.tx.[j-1] = '\013' && i>0 ->
1381   _____c.cpos <- (i-1,j-1);
1382   (* if find newline just go back *)
1383   _____| (i,j) when c.stx = 0 && i<=0 -> c.sty <-
1384   _____pop-y (); c.cpos <- (0,j-1);
1385   (* if first position and first line scroll back *)
1386   _____| (i,j) when c.stx <=0 -> c.cpos <- (i-1,j-1);
1387   (* if first position and not first line go back *)
1388   _____| (i,j) -> c.cpos <- (i,j-1);
1389 in
1390 (* goto the position on the text *)
1391 let rec goto_position c position=
1392   _____match c.cpos with
1393   _____ (i, j) when j> position -> ()
1394   _____| _ -> stp c; goto_position c position
1395 in
1396 (* goto the end of the document *)
1397 let goto_end_of_doc c= reset_editor c; goto_position c (last_position c)
1398 in
1399 (* goto n positions back *)
1400 let rec goto_pos_bk c pos =
1401   _____if (c.stx <=pos) then ()
1402   _____else (bkstep c; c.stx <-
1403   _____c.stx -1; goto_pos_bk c pos)
```

```
1402 in
1403 (* goto first position of the line *)
1404 let rec goto_first_pos_of_line c =
1405   _____c.cpos <- ((fst c.cpos), (snd c.cpos - c.stx)); c.stx <- 0;
1406 in
1407 (* go previous line step 1 *)
1408 let upstp c =
1409   _____sts.bkpx <- c.stx;
1410   _____c.cpos <- ((fst c.cpos), (snd c.cpos - c.stx)); c.stx <- 0;
1411   _____bkstep c;
1412   _____sts.virtual_char <- 31;
1413 in
1414 (* go previous line step 2 *)
1415 let upstp_2 c =
1416   _____goto_pos_bk c sts.bkpx;
1417 in
1418 (* goto on a position in the line *)
1419 let rec goto_pos c pos = match c.cpos with
1420   _____ (i, j) when j >= (String.length c.tx) - 2 -> ()
1421   _____| (i, j) when c.stx >= glb.chars_per_line &&
1422   _____ pos >= glb.chars_per_line -> ();
1423   _____| (i, j) when c.stx >= glb.chars_per_line -> ()
1424   _____| (i, j) when c.stx >= pos -> ()
1425   _____| (i, j) when c.tx[(snd c.cpos)] = '\013' -> ()
1426   _____| _ -> stp c; goto_pos c pos
1427 in
1428 (* goto end of the line *)
1429 let goto_end_of_line c =
1430   _____( if c.stx = glb.lines_per_page then () );
1431   _____goto_pos c ( glb.chars_per_line );
1432 in
1433 let rec nxt_line_loop c =
1434   _____if c.stx = 0 || (snd c.cpos) > (String.length c.tx) - 2 then ()
1435   _____else (stp c; nxt_line_loop c)
```

```
1436 in
1437 (* goto first position of the next line *)
1438 let nxt_line c = stp c; nxt_line_loop c;
1439 in
1440 (* go down one line on the same position *)
1441 let dwnstp c =
1442     sts.bkpx <- c.stx; nxt_line c; goto_pos c sts.bkpx;
1443 in
1444 (* insert a char on the text if its posible *)
1445 let rec over_char ch c = match c.cpos with
1446     | (i,j) when j < (String.length c.tx) - 1 ->
1447         let x= c.tx in
1448             ( if (mask_fun c.mask) j then x.[j] <- ch; stp c)
1449     | _ -> ()

1451 in
1452 let ins_txt c txt=
1453     for i= 0 to String.length txt - 1 do
1454         over_char txt.[i] c
1455     done
1456 in
1457 let del_char c = match c.cpos with
1458     _____(i,j) -> if (mask_fun c.mask) j then c.tx.[j] <- '#'
1459 in
1460 (* delete j caracteres from the cursor *)
1461 let rec del_them j c=
1462     if j < 1 then ()
1463     else ( del_char c; del_them (j-1) c )
1464 in
1465 (* delete from mark to cursor *)
1466 let del_txt c = match c.cpos with
1467     _____(i, j) when j > c.mark -> ()
1468     _____ | (i, j) -> del_them (c.mark - j) c
1469 in
```

```

1470 let rec find_to j n str= match n with
1471     | t when t < (String.length str) - 2 &&
1472     | str.[t] = '\009' && str.[t+1]= ' ' ->
1473     | t - j
1474     | t when t < (String.length str) - 2 -> find_to j (n+1) str
1475     | t -> (String.length str) - 2 - j
1476 in
1477 (* backstep on a field text *)
1478 let bkstep_field c =
1479     let rec bkstep_field_loop c =
1480         _____match c.tx.[snd c.cpos] with
1481         | _____ x when x='#' || x='@' -> bkstep c;
1482         | _____ -> stp c
1483         _____in match (c.tx.[ snd c.cpos]) with
1484         | _____x when x='#' || x='@'
1485         -> bkstep_field_loop c
1486         | _____| x when (int_of_char x)=13 || x = ' ' -> bkstep c;
1487         | _____| -> ()
1488     in
1489     _____
1490     (*and edTxt= ref {tx= ""; cpos= (0,0); stx=0; sty=0 } *) in
1491     (* Reset the cursor on the display *)
1492     let reset_display c = c.cpos <- (0, c.sty); c.stx <- 0;
1493     in
1494     (* put the cursor on the mouse position *)
1495     let mouse_point c =
1496         let rec loop i =
1497             _____if i<=0 then ()
1498             _____else (dwnstp c; loop (i-1) ) in
1499         let (x,y)= glb.mouse_position in
1500         let (cx, cy)= (min glb.chars_per_line ((x-rc.x-3+caractere_space)/tw),

```

```

1501   min glb.lines_per_page ((rc.y+rc.h-3+th-y)/th)) in
1502   reset_display c;
1503   loop (cy-1);
1504   goto_first_pos_of_line c ; goto_pos c cx;
1505   (* put the cursor at the cx position *)
1506   in
1507   let set_state cpos gcor i=
1508     sts.etx <- cpos;
1509     contour := (get_color gcor);
1510     if i then ( sts.etx.sty <- 0; sts.etx.stx <- 0; sts.sty_stk <- [] )
1511   in let (size_x , size_y)= (Graphics.size_x(), Graphics.size_y()) in
1512   let ctrl_x= make_line_input 20
1513             (mkrec 25 (rc.y - 2*th) (size_x - 50) (2*th))
1514   in
1515   let process_esc k c= match c.cpos with
1516   _____ (i, j) when k= ' ' -> c.mark <- j
1517   _____| (i, j) when k= 'w' -> push_circular
1518   _____( String.sub c.tx (min c.mark j)
1519             (abs (j- c.mark)))
1520   _____| (i, j) when k= 'a' -> reset_editor c
1521   _____| (i, j) when k= 'e' -> goto_end_of_doc c
1522   _____| (i, j) -> ()
1523   and tg= tag id rc
1524   (* Control a virtual keypress *)
1525   and pop_virtualchar () =
1526     let c= sts.virtual_char in ( sts.virtual_char <- 0; c )
1527   in
1528   (* Control of mini buttons in the window editor *)
1529   (* position of the mini buttons *)
1530   let mini_rc = {x= rc.x+rc.w-15; y= rc.y+rc.h-15; w= 15; h= 15 }
1531   and scroll_up_bt = {x= rc.x+rc.w-15; y= rc.y+rc.h-40; w= 15; h= 15 }
1532   and scroll_down_bt = {x= rc.x+rc.w-15; y= rc.y; w= 15; h= 15 }
1533   in
1534   (* draw of the mini button *)

```

```
1535 let draw_mini_buttons () =
1536     Graphics.draw_rect mini_rc.x mini_rc.y mini_rc.w mini_rc.h;
1537     Graphics.draw_rect scroll_up_bt.x scroll_up_bt.y
1538     scroll_up_bt.w scroll_up_bt.h;
1539     Graphics.draw_rect scroll_down_bt.x scroll_down_bt.y
1540     scroll_down_bt.w scroll_down_bt.h;
1541     Graphics.moveto (mini_rc.x + 3) mini_rc.y; Graphics.draw_char 'X';
1542     Graphics.moveto (scroll_up_bt.x+1) (scroll_up_bt.y+1);
1543     Graphics.lineto (scroll_up_bt.x+scroll_up_bt.w/2)
1544     (scroll_up_bt.y+scroll_up_bt.h/2+4);
1545     Graphics.lineto (scroll_up_bt.x+scroll_up_bt.w-1)
1546     (scroll_up_bt.y);
1547     Graphics.moveto (scroll_down_bt.x)
1548     (scroll_down_bt.y+scroll_down_bt.h);
1549     Graphics.lineto (scroll_down_bt.x+scroll_down_bt.w/2+1)
1550     (scroll_down_bt.y+scroll_down_bt.h/2-4);
1551     Graphics.lineto (scroll_down_bt.x+scroll_down_bt.w)
1552     (scroll_down_bt.y+scroll_down_bt.h); in

1554 let
1555     draw_editor ()=
1556     Graphics.auto_synchronize false;
1557     let tg= tag id rc in
1558     draw_box contour.contents 2 tg;
1559     Graphics.set_color background_mask;
1560     Graphics.fill_rect rc.x rc.y rc.w rc.h;
1561     Graphics.set_color contour.contents;
1562     Graphics.draw_rect rc.x rc.y rc.w rc.h;
1563     draw_mini_buttons ();
1564     Graphics.set_color Graphics.black;
1565     draw_string (tg.x+6) (tg.y+ (tg.h-th)/2) (pilha.(id));
1566     drawMask (rc.x+3) (rc.y + rc.h - th- 3) sts.etx;
1567     ctrl_x.draw_line ();
1568     Graphics.auto_synchronize true
```

```
1569 and
1570   in_muride () =
1571     if mouse_in_window (Graphics.mouse_pos()) tg
1572     then
1573       (sts.insertions <- true; char_of_int 21)
1574     else (sts.insertions <- false; char_of_int 0)
1575 and
1576   in_char ()=
1577     if sts.virtual_char <> 0 then char_of_int (pop_virtualchar () )
1578   (* if there are any key on the virtual keybord then return the top one *)
1579   else begin
1580     let ev= Graphics.wait_next_event [Graphics.Button_down;
1581     Graphics.Key_pressed] in
1582   (* wait for a key pressed or a mouse button down *)
1583   glb.mouse_position <- Graphics.mouse_pos();
1584   (* storage the position of the mouse *)
1585   match ev with
1586     event when event.Graphics.keypressed -> ev.Graphics.key
1587   (* if a key is pressed then return this key *)
1588   | event when (mouse_in_window glb.mouse_position tg) ->
1589     sts.insertions <- true; char_of_int 0
1590   (* if click on the tag active the editor id *)
1591   | event when (mouse_in_window glb.mouse_position
1592   scroll_up_bt) ->
1593     sts.insertions <- true; char_of_int 21
1594   (* if click on scroll down button do it *)
1595   | event when (mouse_in_window glb.mouse_position
1596   scroll_down_bt) ->
1597     sts.insertions <- true; char_of_int 09
1598   (* if click on scroll up button do it *)
1599   | event when (mouse_in_window glb.mouse_position
1600   mini_rc) ->
1601     char_of_int 20
1602   (* if click on the mini-button then return a ascii 20 *)
```

```
1600  _____| event when (mouse_in_window glb.mouse_position rc) ->
1601  _____sts.insertions <- true; char_of_int 28
1602  (* if mouse in the editor winodw then move the cursor
1603  to it position on keypressed ascii 28 *)
1604  _____| - -> sts.insertions <- false; char_of_int 0
1605  end
1606  in
1607  set_state {tx= ""; cpos= (0,0); stx=0; sty=0; mark=0; mask= ""}
1608  (CLR Graphics.red) true;

1610  ctrl_x.state [' -'];
1611  draw_editor ();

1613  (* This is the process event! *)
1614  let text_input ()=
1615  let lines = if oldStuff.(id) = "" then the_text
1616  else oldStuff.(id) and
1617  mascara= if mskStuff.(id)= "" then
1618  (String.copy the_text)
1619  else mskStuff.(id) in
1620  let
1621  c= createMaskText lines (*if lines="" then
1622  (String.make 500 ' ') else lines*)
1623  mascara and
1624  ch= ref (char_of_int 0) in
1625  (
1626  c.stx <- 0;
1627  c.sty <- 0;
1628  ch :=in_muride ();
1629  while sts.insertions do
1630  if !ch<> char_of_int 21
1631  then ch := in_char () else (ch:= char_of_int 0);

1632  (*mainkeys*)
```

```

1633      ( match (int_of_char !ch) with
1634  _____ 01 (*Ctrl-a Goto first position of the line *) ->
1635  _____ goto_first_pos_of_line c
1636  _____| 02 (*Ctrl-b - Backstep *) -> bkstep c
1637  _____| 03 (*Ctrl-c *) -> ()
1638  _____| 04 (*Ctrl-d Backspace *) -> del_char c
1639  _____| 05 (*Ctrl-e goto end of line*) -> goto_end_of_line c
1640  _____| 06 (*Ctrl-f *)-> stp c
1641  _____| 07 (*Ctrl-g *) -> ()
1642  _____| 08 (*Ctrl-h & Backspace *) -> if (snd c.cpos)>0 then
1643  _____| _____( bkstep c; if sts.bkerase then del_char c)
1644  _____| 09 (*Ctrl-i *) -> scroll c; sts.virtual_char <- 14;
1645  _____| 10 (*Ctrl-j insert a line break
      *) -> over_char '\n' c
1646  _____| 11 (*Ctrl-l *) -> ()
1647  _____| 12 (*Ctrl-m *) -> ()
1648  _____| 13 (*ENTER *) -> nxt_line c
1649  _____| 14 (*Ctrl-n Move to next line *) -> dwnstp c;

1651  _____| 15 (*Ctrl-o *) -> ()
1652  _____| 16 (*Ctrl-P Move to previous Line Step-1*) ->
      upstp c
1653  _____| 31 (*Ctrl-P Move to previous Line Step-2*) -> upstp_2 c
1654  _____| 17 (*Ctrl-q Toggle between bkstep and bkerase *) ->
1655  _____| sts.bkerase <- not sts.bkerase
1656  _____| 18 (*Ctrl-r search text backward *) ->
1657  _____| ctrl_x.line_input 2 id c;
1658  _____| reset_editor c;
1659  _____| goto_position c (glb.search_pos -2);
1660  _____| 19 (*Ctrl-s search text forward *) ->
1661  _____| ctrl_x.line_input 1 id c;
1662  _____| reset_editor c;
1663  _____| goto_position c (glb.search_pos -2)
1664  _____| 20 (*Ctrl-t: Applica função *) ->

```

```

1665 _____sts.ctx_save <- c.tx;
1666 _____sts.cmk_save <- c.mask;
1667 _____reset_editor c;
1668 _____c.tx <- fn (pp c.mask c.tx)
1669 _____| 21 (*Ctrl-u *) -> scroll_back c;sts.virtual_char <- 16;
1670 _____| 22 (*Ctrl-v *) -> ctx_swap c
1671 _____| 23 (*Ctrl-w Erase marked area *) ->
1672 _____process_esc 'w' c;del_txt c
1673 _____| 24 (*Ctrl-x Execute function *) ->
1674 _____ctrl_x.state ['C';'n';'t';'r';'l';'-';'X'];
1675     ctrl_x.draw_line ();
1676     oldStuff.(id) <- c.tx;
1677     ctrl_x.line_input 0 id c;
1678     c.tx <- oldStuff.(id);
1679     (* oldStuff.(id) <- c.mask; *)
1680     reset_editor c;
1681     draw_editor()
1682 _____| 25 when getop ()<>"" -> ins_txt c (getop ())
1683 _____| 26 (*Ctrl-z *) -> sts.insertions <- false
1684 _____| 27 (* Esc *) -> process_esc (Graphics.read_key()) c
1685 _____| 28             -> mouse_point c;bkstep_field c;
1686 _____| x when ch.contents >= ' '
1687 _____|   (* Insert char *) ->
1687 _____over_char ch.contents c  (**)
1688 _____| - -> ());

1690         set_state c (CLR Graphics.blue) false;
1691         draw_editor();
1692     done;
1693         if !contour= Graphics.blue then (set_state c
1694         (CLR Graphics.red) true; draw_editor())
1695     else ();
1696     sts.insertions <- false;
1697     c.stx <- 0;

```

```

1698      c.sty <- 0;
1699      sts.sty_stk <- []
1700  ) in oldStuff.(id) <- ed_lines;
1701      mskStuff.(id) <- String.copy ed_lines;
1702      eds.(id) <- {process_event=text_input;
1703      st= set_state; draw_self= draw_editor};
1704      glb.refresh_list <- eds.(id)::glb.refresh_list;
1705      eds.(id)
1706  ;;

1708  let mk_entry ____?(txt="tot")
1709      ____      ?(fn= fun s -> s)
1710      ____      ?(pp= get_fields) nm =
1711  let (cw, ch)= resize_char (Graphics.text_size "w") in
1712  let (size_x , size_y)= (Graphics.size_x(), Graphics.size_y()) in
1713  let rc= (25, size_y - (winHeight+6)*ch, size_x - 50, 100) in
1714      glb.chars_per_line <- (size_x - 100) / cw;
1715      make_entry ~the_text:(String.copy txt)
      ~fn:fn ~pp:pp nm rc;;

1717  let the_mask xs= String.concat "\013" xs;;

1720  (* read a file with the image in marshall format *)
1721  let rdMsh file_nm=
1722      let chan= open_in_bin file_nm in
1723      let m= Marshal.from_channel chan in
1724      let sm= Array.length m.(0) and sn= Array.length m in
1725      (sm, sn, m)
1726  ;;

1728  let make_gbutton img1 img2 fn txt (x0, y0)=
1729  let (w, h, mat1)= rdMsh img1 and
1730      (_, _, mat2)= rdMsh img2 in

```

```
1731 let image1= Graphics.make_image mat1 and
1732     image2= Graphics.make_image mat2 and
1733     (char_w, char_h)= Graphics.text_size txt in
1734 let
1735     rc = mkrec x0 y0 w (h+char_h) and
1736     pressed = ref false and
1737     stat= ref {cpos= (0,0); tx=txt; stx=0;
1738     sty=0; mark=0; mask= "" } and
1739     contour= ref image1 in
1740 let draw ()=
1741     Graphics.auto_synchronize false;
1742     Graphics.draw_image !contour rc.x (rc.y+char_h);
1743     draw_string (rc.x+(w-char_w)/2) rc.y txt;
1744     Graphics.draw_rect rc.x rc.y rc.w rc.h;

1746     Graphics.auto_synchronize true
1747 and set_state c gcor i=
1748     contour:= (get_img image1 gcor) in
1749 draw ();
1750 let button_input ()=
1751     if (click_in_window rc) && not
1752     pressed.contents then (fn(); pressed := true;
1753     set_state !stat (IMG image2) false; draw());
1754     if not(Graphics.button_down()) &&
1755     pressed.contents then (pressed := false;
1756     set_state !stat (IMG image1) false; draw ())
1757     in
1758         let theButton= {process_event=button_input;
1759         st= set_state; draw_self= draw}
1760         in ( glb.refresh_list <- theButton::glb.refresh_list;
1761             theButton)
1762 ;;

1764 let gbutton i img1 img2 fs nm=
```

```
1765 let (cw, ch)= Graphics.text_size "w" in
1766 let (w, h)= (12*cw, 2*ch) in
1767 let (size_x, size_y)= (Graphics.size_x(), Graphics.size_y()) in
1768 let rc= (25+ i* w, size_y-(winHeight+8)*ch - 2*h -10) in
1769     make_gbutton img1 img2 fs nm rc
1770 ;;

1772 (* returns numbers in float format in a string*)
1773 let rec rdThem s=
1774     try
1775         match Stream.next s with
1776         | Genlex.Int x -> (float_of_int x)::(rdThem s)
1777         | Genlex.Float x -> x::(rdThem s)
1778         | _ -> rdThem s
1779     with
1780     _ -> []
1781 ;;

1783 (* get a list of float using ocamllex *)
1784 let getFloatList s=
1785     let lex= Genlex.make_lexer [] in
1786     let strm= lex (Stream.of_string s) in
1787     rdThem strm
1788 ;;

1789 (* returns numbers in float format in a string*)
1790 let rec rdstr s=
1791     try
1792         match Stream.next s with
1793         | Genlex.Int x -> (string_of_int x)::(rdstr s)
1794         | Genlex.Float x -> (string_of_float x)::(rdstr s)
1795         | Genlex.Ident x -> x::(rdstr s)
1796         | Genlex.Kwd x -> x::(rdstr s)
1797         | Genlex.String x -> x::(rdstr s)
1798         | Genlex.Char x -> (String.make 1 x)::(rdstr s)
```

```
1799   with
1800       - -> []
1801   ;;

1803 (* get a list of float using ocamllex *)
1804 let getStringList s=
1805     let lex= Genlex.make_lexer
1806     [ ". "; ", "; "; "; "? "; "! "; "( "; ") "; "+ "; "* "; "- "; "/" ] in
1807     let strm= lex (Stream.of_string s) in
1808         rdstr strm
1809   ;;

1811 let start_mask x=
1812     String.length x > 1 && x.[0]= '\\\ '
1813   ;;

1815 let get_a_mask ms=
1816     let rec loop s acc= match s with
1817         [] -> (acc , s)
1818         | (x::xs) when start_mask x -> (x::acc , xs)
1819         | (x::xs) -> loop xs (x::acc)
1820     in
1821         loop ms []
1822   ;;

1824 let rec prt_list s= match s with
1825     [] -> print_endline "Rest"; flush stdout
1826     | x::xs -> print_endline x; prt_list xs
1827   ;;

1829 let get_masks ms=
1830     let rec loop s acc = match (get_a_mask s) with
1831         ([], []) -> []
1832         | (xs, []) -> (the_mask xs)::acc
```

```
1833         | (xs, rest) -> loop rest ((the_mask xs)::acc)
1834     in
1835         loop ms []
1836     ;;

1839 let readMaskList fileName=
1840     let chan= open_in fileName and
1841         lines= ref [] in
1842     try
1843         while true do
1844             lines := (input_line chan)::lines.contents
1845         done;

1847         (get_masks lines.contents)
1848     with
1849         any -> get_masks lines.contents
1850     ;;

1852 let mask_circle s=
1853     let ms= ref s in
1854     (fun () ->
1855         match ms.contents with
1856             [] when s= [] -> "no_mask_available"
1857             | [] -> ms := List.tl s; String.copy (List.hd s)
1858             | (x::xs) -> ms := xs; String.copy x
1859     )
1860     ;;
```

# Anexo II

Listagem do programa exemplo "nw.ml"

```
1  (***** test framework *****)

3  (* Example routine to modify the text to uppercase *)
4  let upc nm ()=
5  _____let xs= Wid.get_contents nm in
6  _____let us= String.uppercase xs in
7  _____Wid.set_contents us nm;;

8  (* Example routine to modify the text to lowercase *)
9  let toLower nm ()=
10 _____let xs= Wid.get_contents nm in
11 _____let us= String.lowercase xs in
12 _____Wid.set_contents us nm;;

14 (* Example routine to save the text to a file *)
15 let cnt= ref 0;;
16 let savit nm ()=
17 _____let s= Wid.get_contents nm and
18 _____chan= open_out (nm^".b"^(string_of_int !cnt)) in
19 _____incr cnt;
20 _____output_string chan s;
21 _____close_out chan;;

23 (* Example routine to calculate the mean of a list of numbers *)
24 let avg s=
25 _____let rec loop nums acc c= match nums with
```

```
26 _____ [] -> acc /. c
27 _____| x::xs -> loop xs (acc +. x) (c +. 1.0) in
28 _____loop s 0.0 0.0;;

30 (* returns just the numbers in float format *)
31 let rec rdThem s=
32   try
33     match Stream.next s with
34     | Genlex.Int x -> (float_of_int x)::(rdThem s)
35     | Genlex.Float x -> x::(rdThem s)
36     | _ -> rdThem s
37   with
38     _ -> []
39   ;;

41 (* get a list of float using ocamllex *)
42 let getFloatList s=
43   let lex= Genlex.make_lexer [] in
44   let strm= lex (Stream.of_string s) in
45     rdThem strm
46   ;;

48 (* process the text and return the text result *)
49 let find_avg s=
50   let xs= getFloatList s in
51     match xs with
52     | [] -> "Empty_list"
53     | nums -> "The_average_is_"^(string_of_float
54       (avg nums));;

56 (* just a shell to process the text with the mask *)
57 let processa s= find_avg s;;
```

```

60 let main () =
61   _____Wid.open_gr "eMacs_like_Editor_-_Gui_Example";
62   _____let msk= Wid.the_mask
63   _____["Custo: _#####\013Valor_Venal: _#####";
64   _____ "Onda_preta: _#####_"] in
65   _____let ed = Wid.mk_edt ~txt:(Wid.nlines 20)
66   _____~fn:(fun x -> (String.uppercase x)) "Editor_1"
        and
67   _____ed1= Wid.mk_edt "Editor_2" and
68   _____ed2 = Wid.mk_entry ~txt:msk ~fn:processa "Editor_3"
        in
69   _____let ext= Wid.exit_button() and
70   _____uppr= Wid.gbutton 0 "um.nss" "tres.nss"
71   _____(upc "Editor_1") "Uppercase" and
72   _____lc= Wid.button 1 (toLower "Editor_1") "Lowercase"
        and
73   _____donothing= Wid.next_button_row 1
74   _____(fun s () -> ()) "Donothing"
75   _____in

77   _____while true do
78   _____Wid.refresh ();
79   _____ed.Wid.process_event();
80   _____ed1.Wid.process_event ();
81   _____ed2.Wid.process_event ();
82   _____uppr.Wid.process_event ();
83   _____lc.Wid.process_event ();
84   _____donothing.Wid.process_event ();

86   _____ext.Wid.process_event ()
87   _____done;
88   _____Wid.close_gr ();;

90 main ();;

```