UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE SEMICONDUTORES, INSTRUMENTOS E FOTÔNICA

AMBIENTE COMPUTACIONAL INTEGRADO DE SUPORTE AO ENSINO DE ENGENHARIA

por: Flávio Araújo Junior

orientador: Prof. Dr. Furio Damiani

Dissertação submetida à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Engenharia Elétrica.

setembro 1993

Na arte como na ciência,
no fazer como no agir,
tudo está em apanhar
claramente um assunto e
tratá-lo de conformidade
com sua natureza.

Goethe

Resumo

O uso de computadores como elemento didático revela-se uma questão de grande importância.

Neste trabalho, fazemos a apresentação e a descrição de um ambiente computacional de auxílio ao ensino de engenharia. Ele possui características especiais, que permitem, por exemplo, a execução de programas de simulação de modelos de fenômenos físicos, a comparação de seus resultados com dados experimentais, o estudo de tópicos com hipertexto e gráficos multidimensionais e a interação com um banco de conhecimento interdisciplinar. Incorpora também meios de comunicação entre usuários e apresenta um mecanismo de controle estatístico de utilização.

Através deste ambiente, o usuário poderá estudar os modelos seguindo um esquema de disciplinas e tópicos e recorrer ao auxílio de um sistema especialista que tem a finalidade de responder às diversas questões a ele encaminhadas. O programa sugere uma linha de estudo, da qual o usuário poderá determinar o ritmo e a seqüência dos assuntos a serem estudados.

Sua interface gráfica constitui-se em fator essencial, tornando a relação programa-usuário mais amigável. Os recursos gráficos utilizados, o esquema de hipertexto que contém, bem como outros elementos de multimídia, evidenciam a preocupação em compor um ambiente computacional onde haja uma interação intensa com o usuário.

A instalação deste programa está direcionada a uma rede de estações de trabalho e procura partilhar o maior número de recursos oferecidos nesta configuração. Seu banco de dados é bastante flexível, possibilitando atualizações e expansões. A implementação, baseada em uma linguagem estruturada, é modular e visa principalmente, a portabilidade e a praticidade de manutenção.

Abstract

In this work, we present and describe a computational environment to give support to engineering teaching. It has special features, allowing, for example, the execution of physical phenomena simulation programs, the comparison between its results and experimental data, the study of topics using hypertext and multidimensional graphics and user interaction with a interdisciplinary data base. It also offers the possibility of communication among its users. It features a tool of detailed statistical measurement of its use.

The program suggests a study strategy using a disciplines and topics structure. The user can further define the rhythm and the sequence of subjects at his will. An expert system aims to answer the questions sent to it, accessing a dynamic knowledge data base.

Its graphical interface (GUI) plays an important role, creating a user-friendly relationship. The graphical resources employed, its hypertext and other multimedia elements add to a computational environment marked by an intensive user interaction.

The program is meant to be used in a workstations network. It trys to share the majority of available resources in this configuration. Its data base has a great flexibility, allowing updating and upgrading. The implementation, based in a structured language, is modular, aiming to portability and to the ease of maintenance.

Agradecimentos

Inicialmente, gostaria de agradecer a toda minha família, em especial ao meu pai, Flávio Araújo, pela compreensão e apoio em todos os momentos de minha vida acadêmica.

Agradeço ao meu orientador, Prof. Dr. Furio Damiani, pela oportunidade oferecida de poder desenvolver um importante trabalho de tese e pela confiança na minha atuação junto à administração da rede de estações de trabalho do departamento.

Sou grato também ao Prof. Dr. Peter Jürgen Tatsch, pela sua essencial colaboração durante todas as fases deste projeto. Ambos os professores me proporcionaram um excelente ambiente para estudo e desenvolvimento de minhas capacidades.

Devo registrar as valiosas contribuições dos amigos Luis Henrique Guilherme Rosalino, na especificação e implementação de alguns módulos do programa, Leandro Dibb, na manipulação de figuras em *PostScript* e Haroldo José Onisto, no trabalho com gráficos.

Por último, não poderia deixar de agradecer à Eliane França e aos demais colegas de departamento pelo agradável tempo de convivência e pelos encontros quase que diários, para um café na cantina do CABS (Centro Acadêmico Bernardo Sayão).

Conteúdo

Lista de Figuras

Capítulo 1

Introdução e Objetivos

Neste capítulo relacionamos as idéias que nos levaram à realização deste trabalho de tese. Apresentamos também o contexto no qual está inserido o projeto computacional que ele descreve e seus objetivos.

1.1 Introdução

O entendimento de um determinado assunto e a busca do seu conhecimento dependem, em grande parte, do grau de motivação individual.

De maneira geral, o ensino dentro dos cursos de graduação em uma universidade, se processa através de aulas teóricas onde está em evidência a figura do professor, que de uma certa forma, é responsável pela motivação dos seus alunos.

No entanto, dependendo do curso, algumas atividades de caráter prático têm papel de destaque. Dentre elas podemos citar a utilização de laboratórios e a realização de seminários [1]. Estas atividades vêm complementar as aulas expositivas e procuram fazer com que o aluno interaja com o seu meio e que ele tenha uma participação mais efetiva no seu processo de formação profissional.

Contudo, estes métodos possuem limitações que vêm implicar numa eficiência relativa e, em algumas vezes, na perda da qualidade do assunto a ser transmitido.

A principal deficiência da aula tradicional está na falta de dinamismo e na relevância subjetiva com que o professor trata determinados assuntos. Como conseqüência pode ocorrer o desinteresse dos alunos pela aula e pelo próprio assunto.

Entretanto, é importante salientar que a troca de informações em uma aula normal vai além do que se encontra nos livros. Na aula o professor transmite muito da sua experiência individual que enriquece o assunto e auxilia em grande parte a sua transmissão.

Na realidade, podemos destacar que numa turma de alunos nem todos gostam dos mesmos assuntos, tampouco com a mesma intensidade. Além disso, o interesse demonstrado por um certo aluno pode estar prejudicado, em algum instante, por motivos alheios ao que se passa dentro de uma sala de aula. Em resumo, podemos dizer que para o aluno, o modelo ideal de ensino seria o de estudar só aquilo que mais lhe interessa e nas horas em que lhe fosse mais conveniente.

Com relação às atividades práticas anteriormente citadas, observamos também uma distorção no grau de aproveitamento dos alunos. Estas atividades ocorrem geralmente em grupos onde o nível de conhecimento de cada aluno nem sempre é respeitado, gerando em determinados momentos, insatisfações.

O conjunto destes métodos, que caracteriza o ensino de massa tradicional, busca a otimização do processo de transmissão de conhecimento, tendo por base a existência de um aluno de nível médio. Este aluno seria capaz de acompanhar todas as aulas e aprender os assuntos na velocidade em que lhe são apresentados [1].

Entretanto, o aluno de nível médio não corresponde à realidade. Enquanto alguns alunos conseguem aprender um determinado assunto com rapidez, outros levam muito mais tempo para isso [1].

Por outro lado, nos dias atuais não é possível atender às necessidades individuais do aluno. A qualidade da informação se encontra prejudicada pela relação professor-alunos, o que inviabiliza a prática de ensino dedicado a apenas um aluno.

Mas a aula tradicional também produz seus efeitos negativos para com o professor que, normalmente, não pode conhecer as dificuldades e os interesses de cada aluno. Este acompanhamento, mais uma vez é feito por uma média suposta, desconsiderando o grau de conhecimento individual.

É fundamental para o professor acompanhar os meios utilizados pelos alunos para o estudo, bem como identificar os pontos críticos que dificultam este entendimento. Também é importante analisar as etapas percorridas pelo aluno para alcançar seus objetivos e o tempo que ele gasta em cada uma delas.

A partir destas informações o professor pode programar melhor sua aula, prática ou teórica, de modo a sanar deficiências anteriores. Pode perceber inclusive, as reais motivações e as fraquezas que cada aluno apresenta.

O uso de computadores como complemento das atividades desenvolvidas em sala de aula tenta diminuir os espaços deixados pelos métodos tradicionais de ensino [1]. Através da utilização de computadores o aluno pode se sentir mais motivado pelo assunto da aula e procurar satifazer suas necessidades de compreensão.

Dessa forma, é o próprio aluno que poderá determinar o ritmo e a seqüência dos assuntos a serem aprendidos. Além disso, relação entre o aluno e o objeto de estudo é muito mais intensa; o aluno tem maior liberdade para fazer suas experências e no número de vezes que lhe convier.

Atualmente, esta utilização representa não só um meio para apresentação de textos e imagens, mas oferece a possibilidade de interação com o usuário. Esta interação é uma propriedade altamente desejada, pois proporciona uma flexibilidade maior no uso dos recursos computacionais.

Neste contexto, o computador é altamente adequado à simulação de experimentos. Vários programas com esta finalidade continuam sendo desenvol-

vidos, ao lado de tantos outros existentes, já amplamente utilizados.

A simulação de modelos de fenômenos físicos através de computadores, auxiliada pelo uso de interfaces cada vez mais próximas da realidade do usuário, constitui uma forma prática e segura para obtenção de resultados. Portanto, o desenvolvimento de um programa de simulação vem acrescentar subsídios importantes dentro do contexto do ensino.

Unindo todas estas características desejáveis, passamos a propor um programa computacional para aumentar o rendimento e complementar as necessidades de quem quer aprender, respeitando a sua individualidade.

Este trabalho vem apresentar o desenvolvimento de um programa para uso em computadores de modo a auxiliar as aulas em cursos em geral. Trata-se de um ambiente que oferece uma série de recursos para a simulação de modelos e que, acima de tudo, procura ser didático, ao contrário dos programas de simulação tradicionais, que limitam-se à apresentação de resultados a partir de um conjunto de informações fornecidas.

O ambiente pretende ser bastante amplo e flexível, observando-se algumas limitações. Apresenta-se de fácil utlização para qualquer usuário, não havendo a necessidade de treinamento prévio.

Através deste programa o aluno terá ao seu dispor a facilidade de interagir com os modelos e de trocar informações com outros usuários. Enquanto isso, o professor poderá verificar de maneira automática, executada pelo programa, o andamento do estudo dos seus alunos e encaminhá-los frente às dúvidas e dificuldades surgidas durante as sessões de simulação.

1.2 Objetivos

Este trabalho de tese tem por finalidade apresentar a especificação e parte da implementação de um ambiente computacional de apoio às aulas de cursos de graduação de diversas áreas do conhecimento. Ele vem servir como substrato para a simulação de modelos de fenômenos físicos.

Capítulo 2

Características do Ambiente

O capítulo descreve as características principais do ambiente, ou seja, quais são os recursos que ele oferece aos seus usuários. Os princípios sobre os quais está baseada a sua implementação e as características relativas à sua estrutura também estão aqui relacionados.

2.1 Características Fundamentais

Voltado às áreas de Ciências Exatas, este ambiente é constituído por uma série de recursos que permitem o estudo e a comparação de modelos que são, por exemplo, utilizados em programas de simulação. A sua principal característica está no fato de ser de uso interativo e o que o diferencia de outros programas de simulação existentes são a sua versatilidade e abrangência, não se limitando apenas a proporcionar um meio para a realização de experiências com modelos, onde o usuário pode fazer medidas, produzir gráficos e imprimir os resultados. Além disso, oferece:

- Um sistema de cadastramento de usuários.
- Controle estatístico de utilização.
- Um sistema exclusivo para troca de mensagens entre usuários.

- A possibilidade do usuário retomar a sua última tarefa a partir do ponto em que houve uma interrupção.
- Um guia de auxílio ao usuário baseado em hipertextos.
- Uma opção de demonstração dos recursos de simulação.
- Um meio de ativação de programas externos ao ambiente.
- Operações de tratamento de informações na base de dados.
- Um editor para a criação de novos módulos de simulação.
- A possibilidade de implantação de um sistema especialista de modo a responder perguntas dos usuários.

Como ambiente de apoio aos cursos de graduação, atende às principais necessidades de alunos e professores [2].

Através dele o aluno tem a possibilidade de:

- Comparar dados de medidas com os resultados teóricos fornecidos pelos modelos.
- Comparar resultados de diferentes modelos.
- Observar limitações dos modelos e sua faixa de validade.
- Observar os efeitos na mudança de parâmetros e variáveis.
- Verificar a sensibilidade de um modelo frente às variações de parâmetros.

A nível do professor o ambiente proporciona:

- Possibilidade de analisar o desempenho dos alunos através de recursos estatísticos.
- Verificar pontos de dificuldade no aprendizado dos alunos.
- Modificar modelos existentes na base de dados.
- Acrescentar novos modelos à base de dados.

2.2 Características a Nível de Programa

As principais idéias que alicerçam a construção do ambiente são [3]:

- Computação pessoal: Os recursos computacionais tornam-se cada vez mais acessíveis e podem ser partilhados por vários usuários. Também apoiado nesta característica, o usuário pode direcionar a utilização do ambiente para atender a todas as suas necessidades.
- Interação: O ambiente computacional deve apresentar uma série de caminhos de modo a fornecer uma resposta imediata a todas as ações dos usuários. Ao mesmo tempo, cada usuário pode trabalhar com o computador da maneira como melhor lhe convém, tornando esta relação mais agradável e dispensando ao mínimo a necessidade de treinamento.
- Gráficos: As pessoas são, na sua maioria, acostumadas ao pensamento visual. O texto pode ser substituído por imagens gráficas tanto quanto possível.
- Programação orientada a objetos: A clareza e a portabilidade de um programa de grande porte são características altamente desejáveis. Além disso, a possibilidade da reutilização de funções anteriormente implementadas e o fato de se trabalhar em um nível mais alto de abstração, em termos de conceitos mais próximos utilizados no dia-a-dia, tornam a programação orientada a objetos a ferramenta mais adequada para a construção de qualquer ambiente computacional.

A instalação deste ambiente está direcionada a uma rede de estações de trabalho. Está implementado em linguagem C seguindo o padrão ANSI e tem por base o sistema operacional UNIX e o sistema de janelas X Window.

Devido às facilidades inerentes de comunicação entre as estações, este ambiente pode ser utilizado por diversos usuários que se encontrem em regiões geográficas distintas.

É constituído fundamentalmente de duas grandes partes: interface gráfica e modelos.

A interface gráfica é responsável pela interpretação das ações desejadas pelo usuário e pela apresentação dos resultados, de maneira simples, de fácil compreensão e ao mesmo tempo completa.

Os modelos caracterizam-se pelos métodos numéricos e equações matemáticas. Estes modelos constituem programas independentes do ambiente de simulação e podem ser acoplados a ele através de recursos que dispõe, próprios para esta tarefa.

Do ponto de vista da programação, o *software* apresentado pode ser dividido nos seguintes módulos, representados na figura 2.1.

- Gerenciador:

  É o principal módulo, sendo responsável pela administração de todo o ambiente. Controla o fluxo de informações, seja entre o usuário e o programa, entre os usuários e também entre programas.

  Cabe ao gerenciador as seguintes tarefas:

  - Verificar permissão de acesso ao ambiente de acordo com o usuário.
  - Permitir o cadastramento de novos usuários.
  - Permitir a troca de mensagens entre usuários que utilizem o ambiente de simulação, sendo portanto, um serviço exclusivo para estes usuários.
  - Realizar consulta no banco de dados.
  - Fazer acompanhamento estatístico de utilização do ambiente.

- Banco de dados:

  Representa o local de armazenamento dos modelos para simulação e de uma biblioteca de referência dos diversos assuntos relacionados aos modelos. Também se encontram armazenados aqui as janelas de apresentação de desenhos dos modelos, suas equações e gráficos.

- Sistema de janelas:

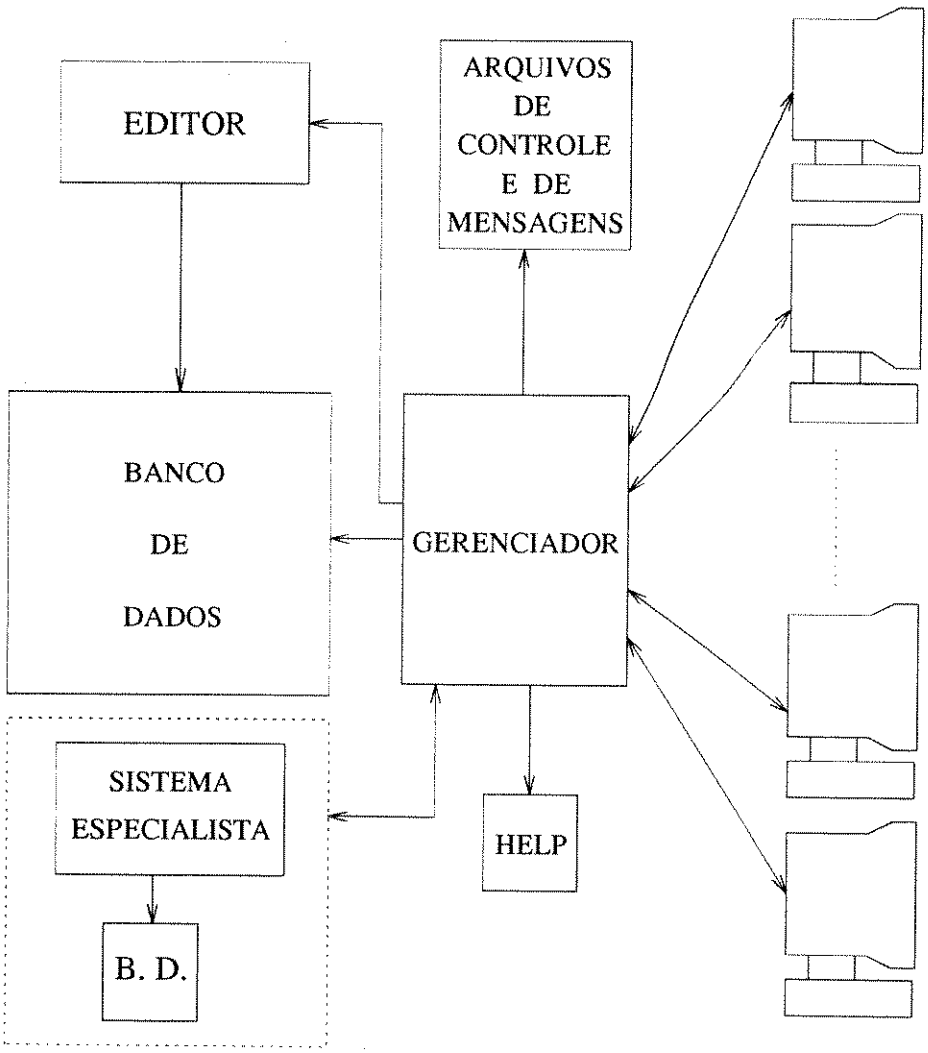

Figura 2.1: Estrutura geral do ambiente.

É a interface programa-usuário. Compõem-se de vários processos disparados pelo gerenciador na estação de trabalho em que se encontra o usuário.

Seu objetivo é fazer com que ele utilize o ambiente de uma maneira interativa e permitir a representação dos modelos e a realização de simulações com o auxílio de importantes recursos gráficos.

- Sistema de auxílio ao usuário em diversos níveis (*Help*):

  Concebido para proporcionar maior facilidade de uso. Está presente nos mais variados níveis da estrutura do ambiente, sendo porém, tratado como um módulo separado para facilitar o acesso às informações. O usuário poderá consultá-lo *on line* ou imprimí-lo como se fosse um manual.

- Editor:

  Tem como objetivo tornar flexível a utilização deste ambiente para as mais diversas áreas do conhecimento. A idéia principal é fazer com que pessoas com pouca familiaridade com o sistema de janelas X Window e que não tenham conhecimento da estrutura deste ambiente de simulação possam utilizar o editor como ferramenta de composição de tópicos de estudo.

  Trata-se de um módulo que irá requerer uma implementação cuidadosa e que, em princípio, será bastante restrito.

- Sistema especialista:

  Seu objetivo neste ambiente é proporcionar ao usuário tirar dúvidas sobre os mais diversos assuntos. Possui um banco de dados próprio constituído de perguntas e respostas. À medida em que é utilizado, este sistema incorpora novas questões ao seu banco de dados de modo a oferecer ao usuário uma fonte crescente de informações.

- Arquivos de controle:

  São arquivos utilizados pelo gerenciador, sempre que necessário, de modo a monitorar o uso do ambiente. Dentre eles podemos citar o arquivo de cadastro de usuários, os arquivos de estatística e os arquivos de mensagens pertencentes ao sistema de correio eletrônico exclusivo.

Capítulo 3

Fundamentos Teóricos

O embasamento teórico que nos preparou para a elaboração deste projeto é descrito neste capítulo. Apresentamos de forma sucinta os conceitos envolvidos, de modo que o leitor possa ter uma noção geral sobre eles.

3.1 Programação Orientada a Objetos

A implementação do ambiente está baseada na programação orientada a objetos que constitui um modo pelo qual os programadores geram funções de forma encapsulada, para serem utilizadas pelos usuários [3]. A ênfase no relacionamento entre programadores e usuários representa a principal característica que separa a programação orientada a objetos da maneira convencional [3].

Este tipo de programação está vinculada ao mecanismo de interconexão, que é o processo de integração funcional de diferentes partes de um programa, possivelmente implementadas por programadores distintos [3].

Os principais elementos representativos da programação orientada a objetos são [4]:

- Objeto: É um conjunto particular de dados e operações que podem acessar estes dados. Um objeto é acionado para executar uma de suas operações

através do envio de uma mensagem informando o que ele deve fazer. O programa que manipula este objeto responde à mensagem, em primeiro lugar escolhendo a operação que a implementa, depois executa esta operação e em seguida retorna o controle ao usuário. O objeto combina a representação flexível de dados com modularidade totalmente controlável.

- Classe: É uma estrutura que contém todos os elementos que compõem uma programação para objeto tais como, os membros que armazenam ítens de dados e as funções-membro que manipulam os dados e se comunicam com o resto do programa. As classes são especialmente importantes porque ajudam a implementar modelos que possibilitam aos programas de computador simular situações complexas da vida real. Elas administram eficientemente a complexidade de grandes programas através da modularização.
- Herança: Não é necessariamente uma característica de uma linguagem orientada a objetos, mas é certamente, uma característica extremamente desejável. Constitui uma maneira de organizar, construir e aproveitar classes reutilizáveis. Torna possível definir um novo *software* sem que seja necessária a compreensão do mecanismo de construção de cada módulo funcional. Trata-se de um mecanismo para criar hierarquia de classes, porém com características adicionais.

É importante salientar que a programação orientada a objetos não requer o uso, necessariamente, de uma linguagem com esta característica.

A utilização de uma linguagem estruturada como C, permite a implantação desta filosofia de programação sem maiores problemas [4].

3.2 Controle de Concorrência

Em um ambiente de multiprogramação, várias transações podem ser executadas concorrentemente [5]. Um caso que merece atenção é quando diversos usuários, ao mesmo tempo, decidem abrir o mesmo arquivo para escrita.

Como conseqüência, este arquivo guardará somente as informações inseridas pelo usuário que o gravou por último em relação aos demais. Este processo gera inconsistência no arquivo, que só garante a informação quando estiver sendo alterado por um único usuário em um dado instante.

Para contornar este problema é necessário que haja um controle da interação entre as transações concorrentes, de modo a evitar que elas destruam a consistência dos arquivos. Este controle é conseguido por meio de uma série de mecanismos que serão chamados esquemas de *controle de concorrência* [5].

Uma solução amplamente utilizada nas redes de estações de trabalho que, por exemplo, usam o sistema operacional UNIX, baseia-se num programa controlador de leitura e escrita em disco, chamado *daemon*. Dessa forma, toda requisição do usuário para acessar os arquivos passa a ser feita diretamente a este programa, de maneira transparente.

O *daemon* é um programa *residente* que é executado numa estação de trabalho escolhida como *servidora*, isto é, na qual estão localizados os arquivos principais de um programa que oferece seus serviços às demais estações conectadas à rede. Estas, por sua vez, comportam-se como subsidiárias e dependentes em relação ao *daemon* da *servidora* [13, 14].

Uma função da biblioteca C chamada **lockf()**, pode ser utilizada como mecanismo de bloqueio e consulta do *status* do arquivo. Ela permite o bloqueio exclusivo de todo o arquivo ou de uma parte dele, impedindo a execução de operações de leitura e escrita [13, 14].

O bloqueio pode ser feito através de um mecanismo rígido ou como uma advertência. No primeiro caso, pode ser compartilhado ou exclusivo. O modo compartilhado dá a outros processos a possibilidade de leitura, restringindo a escrita ao processo bloqueador. Já no modo exclusivo, esses processos não têm nem mesmo a permissão para fazer a leitura no arquivo bloqueado.

Enquanto isso, o esquema de advertência modifica apenas o *status* do arquivo e permite com que outros processos possam ler e escrever, deixando o controle das operações para algum programa que faça o gerenciamento destas

informações.

Esta função trabalha em conjunto com o *daemon* **lockd** que serve a rede de estações. É ele que processa as requisições que são enviadas localmente ou remotamente por outro *daemon*. Também repassa as requisições dos dados remotos para o *daemon* local da estação servidora [13, 14].

3.3 Banco de Dados Distribuído

Um sistema de banco de dados distribuído consiste num conjunto de locais de armazenamento, cada um dos quais podendo participar na execução de transações que, por sua vez, acessam dados em um ou vários destes locais. Portanto, cada local mantém um banco de dados próprio e está apto a processar transações locais ou globais [5].

A correta distribuição dos dados é sem dúvida, uma das características mais importantes para o bom funcionamento deste modelo de armazenamento.

Existem diversas razões para se construir um sistema de dados distribuído [5]:

- Compartilhamento de dados e controle distribuído: Se um número de diferentes locais são interligados, então um usuário em um determinado local pode acessar dados que estão disponíveis em outro. Entretanto, cada local detém um certo controle sobre os dados que estão ali armazenados. Há um administrador global do banco de dados para todo o sistema mas, uma parte destas responsabilidades é delegada ao administrador local.
- Confiabilidade e disponibilidade: No caso em que um local apresenta falhas, os demais podem continuar operando normalmente. Desta forma, o não funcionamento de um local não implica necessariamente na paralisação do sistema.
- Aceleração do processamento de consulta: As consultas podem ser executadas em paralelo por diversos locais. Isto permite rápido processamento

de consulta por parte do usuário.

Todavia, ao lado destes benefícios, observam-se algumas desvantagens:

- Custo do desenvolvimento de *software*: Torna-se mais complicado implementar um sistema de banco de dados distribuído e conseqüentemente, mais caro.
- Maior potencial para erro: As transações ocorrem em paralelo, em diversos locais, aumentando a possibilidade de haver conflitos na troca de informações. A coordenação de todo o sistema envolve uma série de cuidados adicionais.
- Aumento de sobrecarga de processamento: A troca de mensagens e a computação adicional exigida para se conseguir coordenação interlocal é uma forma de sobrecarga.

Existem diversas políticas relativas à implantação de um banco de dados distribuído, dentre as quais destacamos a *replicação* e a *fragmentação* [5].

Na replicação o sistema mantém diversas cópias das informações, colocadas em locais diferentes, enquanto que na fragmentação as informações são divididas em grupos, armazenados entre os vários locais existentes.

Chamamos de *transparência de rede* ao ocultamento de detalhes de distribuição de dados numa rede de computadores. Ela está relacionada com a questão da autonomia local. Neste caso os usuários podem ignorar os detalhes do projeto do sistema distribuído e esta autonomia permite com que o projetista ou administrador local seja independente do sistema de distribuição [5].

3.4 Modularidade e a Linguagem C

A vantagem mais evidente da modularidade é que ela dá ao programador a habilidade para lidar com grandes programas. A linguagem C representa

bem este aspecto dado que a função é o seu módulo mais básico [4].

C oferece algumas ferramentas individuais para o projeto de programação estruturada através da modularização. Podemos criar vários módulos, manipulando as declarações de classe de armazenamento tanto de variáveis como de funções e usando a capacidade C de fazer a compilação separada. Assim, as definições de funções se encontram em muitos arquivos-fonte diferentes, passando-se a compilar e depurar cada arquivo independentemente dos outros e combinando-os somente na fase de *linkedição*.

Os arquivos independentes podem, através das regras de escopo de C criar regiões dentro do programa que mantenham suas próprias variáveis internas e funções, bem como elos com o resto do programa. Estas funções e variáveis podem ser vistas como uma interface que permite o acesso a estas regiões, porém limitado e controlado pelo programador. A habilidade para criar estes objetos de acesso restrito é muitas vezes conhecida como *ocultação* ou *abstração de dados* e é uma técnica útil para controlar a complexidade inerente dos projetos de programação [4].

Representando um valor e definindo operações específicas para manipulá-lo, cria-se uma espécie de *objeto* que espelha a realidade que está se tentando implementar. Os elementos-chave que definem um *objeto* são [4]:

- Uma estrutura-objeto.
- As operações que manipulam esta estrutura.

É justamente através desta representação, que obtemos através da linguagem C, o modelamento para a programação orientada a objetos, de acordo com o primeiro ítem deste capítulo.

A idéia principal é fazer com que os detalhes de representação e manipulação dos dados fiquem ocultos do usuário e de outros programadores que posteriormente, venham a modificar partes do programa ou então desejam utilizar alguns destes módulos em outros programas. Combinamos então a abstração

de dados e a modularidade, de modo a formar uma *unidade* que pode ser conectada a um programa, sem nos preocuparmos, a partir daí, com a maneira através da qual ela tenha sido implementada. Em resumo, cada objeto é um módulo e uma parte independente do programa [4].

3.5 Estrutura das Interfaces Gráficas

Apesar de interfaces que utilizam ícones e programação orientada a objetos serem usualmente discutidas como tópicos relacionados entre si, esta relação é, na verdade, indireta e não necessariamente óbvia. Uma interface baseada em ícones não necessita ser construída através de uma linguagem orientada a objetos e por conseguinte, esta não precisa ser utilizada em um terminal com alta resolução gráfica [3].

As aplicações para o sistema operacional UNIX são baseadas na abstração unidimensional, ou seja, na linha de comando. Esta abstração tem o seu papel de destaque nas tarefas do dia-a-dia de utilização dos dispositivos ligados ao computador. No entanto ela é pobre no que se refere à transmissão de informação às pessoas [3].

Por outro lado, os novos programas estão baseados na abstração bidimensional, como pontos, retas e imagens. Isto é implementado por duas camadas arquiteturais [3]:

- Nível de Aplicação:

  Implementa a funcionalidade da aplicação, desconsiderando de que maneira ela será apresentada ao usuário. Os componentes deste nível são chamados de modelos pelo fato da funcionalidade de muitas aplicações estarem representando alguns aspectos da realidade. Os modelos nunca armazenam qualquer informação sobre a interface com o usuário, uma vez que a camada de apresentação pode ser modificada a qualquer momento. Estes modelos são referenciados através de visões.

- Nível de Apresentação:

  É responsável pelo interfaceamento de uma aplicação para um conjunto de usuários. Os componentes no nível de apresentação têm o nome de visões, cada qual implementando uma interface com o usuário para um modelo específico no nível de aplicação.

  As visões fazem o interfaceamento de um modelo com o usuário através de um dispositivo, tal como um terminal gráfico e um *mouse*. A visão referencia o modelo para obter os dados, formata estes dados, mostra-os ao usuário, recebe comandos deste e comanda o modelo para a próxima tarefa. Os modelos são completamente passivos e nunca armazenam informação que poderia torná-los dependentes de uma camada de apresentação particular.

Acima destas camadas temos a apresentação destas visões, que vêm formar o conjunto da interface gráfica utilizada pelo usuário. Como resultado, obtemos uma série de imagens na tela do computador.

Esta estrutura está representada na figura 3.1.

3.6 Sistema X Window

O sistema X Window, ou simplesmente X, fornece uma hierarquia de janelas que podem ser livremente manipuladas e suporta gráficos de alta *performance*, independentes de dispositivos [7].

X é um substrato sobre o qual praticamente qualquer estilo de interface com o usuário, que necessite de um conjunto bastante flexível de rotinas, pode ser projetado. Este sistema é baseado em um protocolo de rede assíncrono, tornando o mecanismo de rede transparente tanto para o usuário final como para o programador [7].

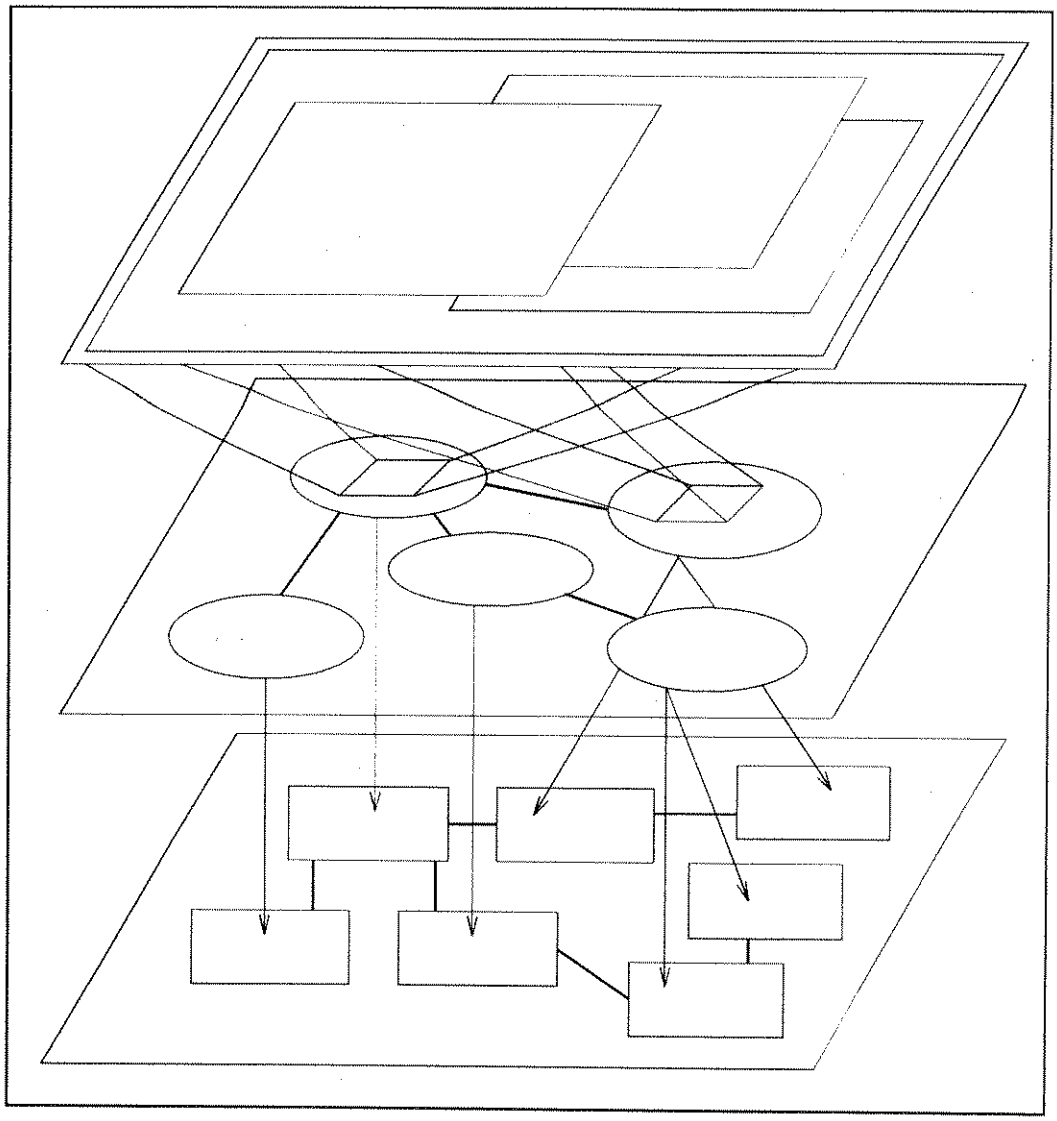

Figura 3.1: Estrutura das interfaces gráficas: a) Nível de aplicação b) Nível de apresentação c) Imagem fornecida ao usuário.

Trata-se, acima de tudo, de um sistema portável e constitui-se num ambiente padrão para aplicações de *software* em estações de trabalho. Proporciona aos usuários uma grande variedade de benefícios podendo-se, por exemplo, obter os recursos gráficos acessando estações remotas da mesma maneira em que são apresentados localmente.

A interface do usuário para com a estação de trabalho resulta na combinação de um *servidor*, de um gerenciador de janelas e do programa de aplicação. A figura 3.2 representa o sistema X Window [7, 8].

O servidor é a camada mais inferior, atuando diretamente ao nível do *hardware*, controlando o *display* e é formado por duas partes, sendo uma dependente e outra independente dos acionadores de dispositivos (*device drivers*). Recebe requisições das camadas superiores e envia-lhes respostas baseado no protocolo X.

O gerenciador de janelas é um programa particular, escrito com as blibliotecas X, com a finalidade de controlar o *layout* das janelas na tela da estação e fornecer uma política para a interface humana.

As aplicações são implementadas usando várias bibliotecas de programação, dentre as quais as mais utilizadas são aquelas escritas em linguagem C. Estas bibliotecas incluem uma interface procedural de baixo nível que usa o protocolo X, chamada Xlib e um conjunto de rotinas de alto nível (*toolkit*) escrito em estilo objeto-orientado, chamado Xt Intrinsics [7, 8].

Xlib contém um grande número de subrotinas e funções que possibilitam a conexão com um servidor de *display*, a criação de janelas e desenhos de gráficos, entre outros.

Toolkits são bibliotecas de programas que são colocados sobre a base do sistema X Window via a interface Xlib. Eles implementam um conjunto de elementos para interface com o usuário, tais como menus e botões, referenciados genericamente como *widgets* e permitem que as aplicações os utilizem através de técnicas de programação orientada a objetos [6, 8].

Durante o projeto de uma aplicação, devemos considerar as vantagens

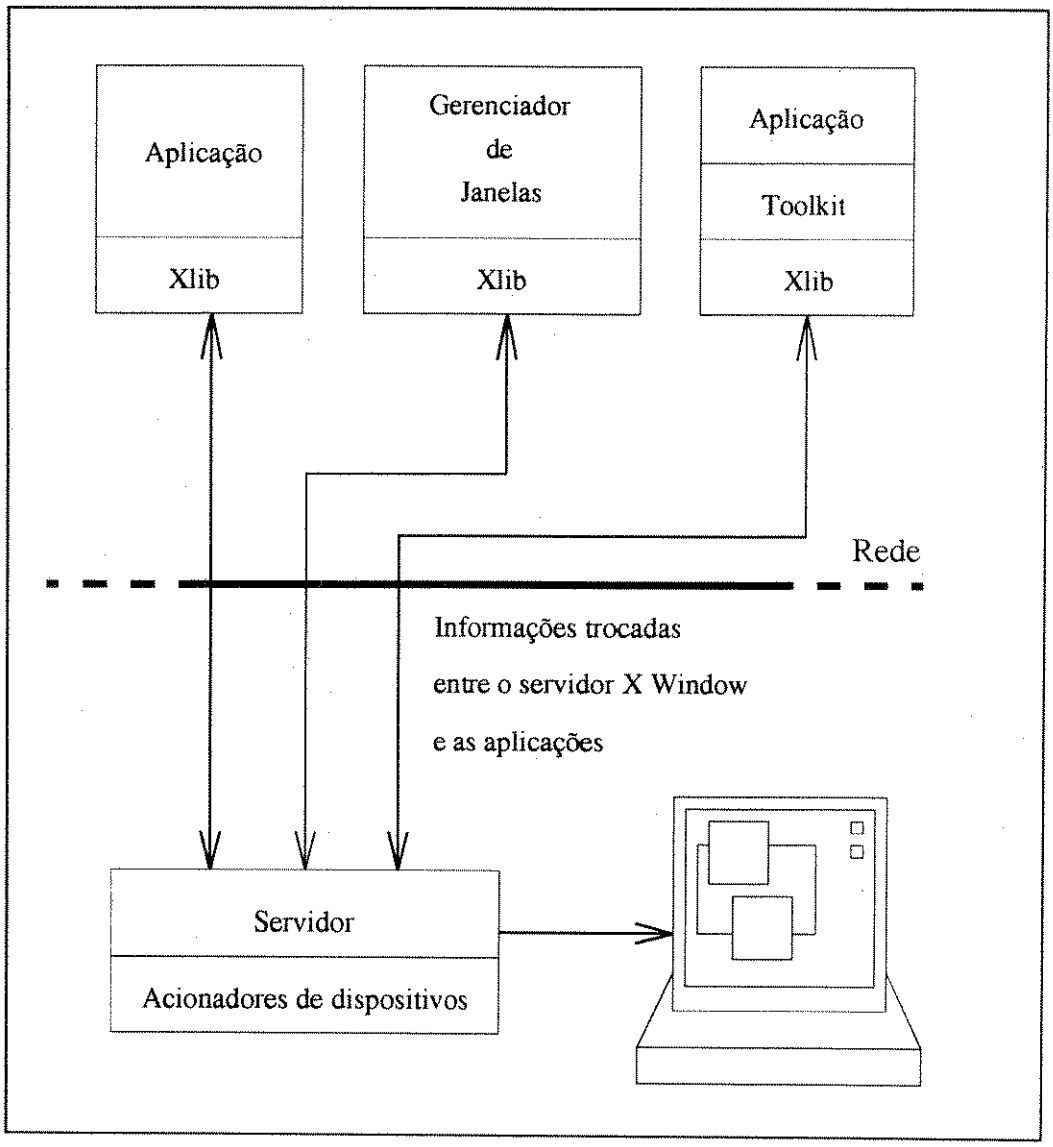

Figura 3.2: Esquema representativo do sistema X Window.

e os inconvenientes da utilização direta do Xlib ou de um *toolkit* particular.

De uma maneira geral, a maioria das aplicações são capazes de utilizar somente os *toolkits*. Entretanto, muitas outras têm a necessidade de usar as funções do Xlib, especialmente as que implementam técnicas de interação gráfica.

Os *toolkits* tornam a programação mais fácil e pode-se tirar vantagem dos padrões já estabelecidos para a interface com o usuário.

No entanto, o programa escrito com um *toolkit* é consideravelmente maior do que o seu equivalente usando Xlib. Além disso, utilizam conceitos altamente abstratos e necessitam de convenções mais restritas na programação.

Capítulo 4

Estrutura do Ambiente

Neste capítulo fazemos a especificação do ambiente. São detalhadas todas as suas características e apresentados os comportamentos de cada parte componente da sua estrutura.

4.1 Filosofia de Implementação

Os objetivos principais que caracterizaram a fase de especificação foram a portabilidade e a facilidade de manutenção do ambiente. Por se tratar de um programa bastante abrangente e que oferece uma quantidade variada de recursos, a sua implementação passou a ser cuidadosamente estudada.

A sua construção foi baseada na filosofia de programação orientada a objetos, uma vez que, a idéia principal era criar um *toolkit* próprio para a interface gráfica com características que possibilitassem maior flexibilidade de uso e atualização futura.

Podemos dizer que do ponto de vista do usuário, um *toolkit* tem a finalidade de padronizar e modularizar as ações interativas, enquanto que para o programador, procura simplificar o projeto e o desenvolvimento de uma aplicação que possua uma ampla interface com o usuário.

Juntamente com estes objetivos e de modo a ter o maior aproveitamento do *hardware* disponível e de mecanismos de programação já existentes, passamos a propor a utilização de hipertextos e de um sistema de arquivos distribuídos neste ambiente.

4.2 Recursos Computacionais Utilizados

A linguagem C padrão ANSI foi escolhida para a implementação, posto que é uma linguagem de uso geral, amplamente utilizada e por conseguinte, se encontra bastante consolidada. O compilador empregado foi o GCC, instalado no sistema operacional UNIX, por se tratar de um compilador que segue o padrão ANSI. Com este conjunto de programas conseguimos reunir a portabilidade da linguagem C, a utilização de um padrão internacional e a facilidade da construção e da obtenção de ferramentas de *software* no sistema operacional UNIX.

Através da escolha principal da linguagem, que oferece um nível de modularização altamente desejado, iniciamos a especificação dos diversos módulos componentes do ambiente. Obtivemos neste caso, uma série de pequenos programas gerados de acordo com a função que representavam.

Outras decisões importantes tomadas durante a implementação deste projeto foram o uso de um mecanismo de controle de concorrência e o emprego de um banco de dados distribuído.

Em relação ao primeiro, a opção adotada foi a aplicação da função **lockf()** nas partes do programa que fazem acesso aos seus arquivos, devido à sua praticidade. Para este caso, inicialmente propusemos a criação de um *daemon* próprio, que teria a vantagem de ser mais flexível quando comparado àquele presente no sistema operacional [13].

Para o banco de dados, o esquema distribuído surgiu como conseqüência da implantação do programa numa configuração em rede de estações de trabalho.

No que se refere ao sistema de interface gráfica, utilizamos o padrão

X Window. Neste caso, passamos a adotar o nível de programação mais inferior, utilizando as rotinas da camada Xlib, em razão de uma maior liberdade quanto à escolha das características visuais do programa.

Muitas das características utilizadas nas janelas de interfaceamento do ambiente foram adaptadas do programa XV, para tratamento de imagens. Trata-se de um programa de domínio público, desenvolvido segundo o padrão X Window, empregando as rotinas do Xlib e que executa sobre o sistema operacional UNIX.

4.3 Localização e Sistema de Arquivos

O ambiente é apresentado para ser instalado em uma rede de estações de trabalho e procura partilhar o maior número de recursos oferecidos nesta configuração. O seu sistema de arquivos está distribuído ao longo da rede, sendo isso no entanto, totalmente transparente ao usuário comum, ao mesmo tempo em que facilita a sua manutenção por parte do administrador da rede.

Seguindo o padrão UNIX para nomes de diretórios, distribuímos os arquivos do ambiente em vários subdiretórios, cada qual com uma determinada finalidade.

Recomendamos a instalação do programa em disco, a partir do diretório **/aces**, derivando dele os seguintes subdiretórios:

- **bin**: Aqui se encontram os arquivos executáveis que compõem todo o ambiente, incluindo o programa principal, chamado **aces**.
- **lib**: Possui os arquivos referentes ao banco de dados. Subdivide-se em:
  - **models**: Contém os arquivos de modelos de fenômenos físicos.
  - **graphs**: Local de armazenagem dos gráficos.
  - **win**: Onde se localizam os arquivos de janelas, escritos usando rotinas do Xlib, representando os diversos elementos de interface com o usuário.

- **man**: Armazena todos os arquivos de texto do ambiente. Está assim subdividido:
  - **help**: Neste diretório se encontram os manuais de utilização do ambiente em geral.
  - **txt**: Diretório onde estão os textos referentes aos modelos implementados.
- **var**: Possui os arquivos de controle do ambiente que caracterizam-se por não ter tamanho fixo. Aqui temos os seguintes subdiretórios:
  - **mail**: Contém os arquivos de mensagens recebidas para cada usuário.
  - **stat**: Onde se encontram os arquivos gerados pelo programa de gerenciamento estatístico.

O ambiente proposto é desenvolvido de modo a aproveitar o mecanismo de *exportação* de diretórios que o sistema UNIX oferece. Trata-se de uma maneira de distribuir os diretórios que compõem este ambiente entre as estações de trabalho que se encontram ligadas em rede.

A implantação de um sistema com informações distribuídas, de um modo geral facilita a sua administração, que pode ser realizada em etapas, não onerando o seu uso, uma vez que o tempo gasto durante o tráfego das informações via rede, na média, pode ser desprezado.

Esta distribuição pode ser feita de dois modos, replicando os diretórios em cada uma das estações ou dividindo-os seguindo um determinado critério.

A replicação oferece maior segurança quando da ocorrência de problemas com uma dada estação, porém com o gasto adicional de espaço em disco. Por outro lado, a segmentação dos diretórios, de forma única para cada estação, reduz ao mínimo a ocupação de disco aumentando, no entanto, a probabilidade de paralisação total do programa na ocorrência de falhas em uma ou mais estações.

Assim, sugerimos que os diretórios **/aces/bin** e **/aces/man/help** fiquem centralizados em uma estação principal e sejam exportados para as sub-

sidiárias. Nestas máquinas estarão localizados os demais diretórios. A figura 4.1 mostra um esquema que representa esta sugestão.

4.4 Gerenciador

É o programa principal propriamente dito e é responsável pela execução de todas as tarefas iniciadas pelos usuários. Este módulo caracteriza-se por um grande *loop*, onde é empregada uma função do Xlib para o tratamento de eventos recebidos pelo ambiente, provenientes do teclado e *mouse* da estação de trabalho.

De acordo com uma ação iniciada pelo usuário, o gerenciador faz a chamada de outro programa sem perder o controle sobre os dispositivos de entrada e saída. Utiliza os recursos oferecidos pelo sistema operacional, enviando arquivos para impressão e invocando programas externos. Também coordena a apresentação das janelas, que formam o conjunto da interface gráfica, na tela da estação.

Além disso, o gerenciador controla diretamente as operações com o banco de dados e supervisiona as funções de leitura e escrita em disco de todos os arquivos manipulados pelo ambiente, seja nos diretórios ocupados pelo programa ou nas *áreas de login* de cada usuário.

4.5 Banco de Dados

O banco de dados a ser utilizado incorpora características do estilo de programação orientada a objetos. É constituído de quatro grandes repositórios de informações:

- Modelos: É o conjunto dos programas que fazem a implementação da parte matemática dos modelos, através da resolução de equações, da aplicação

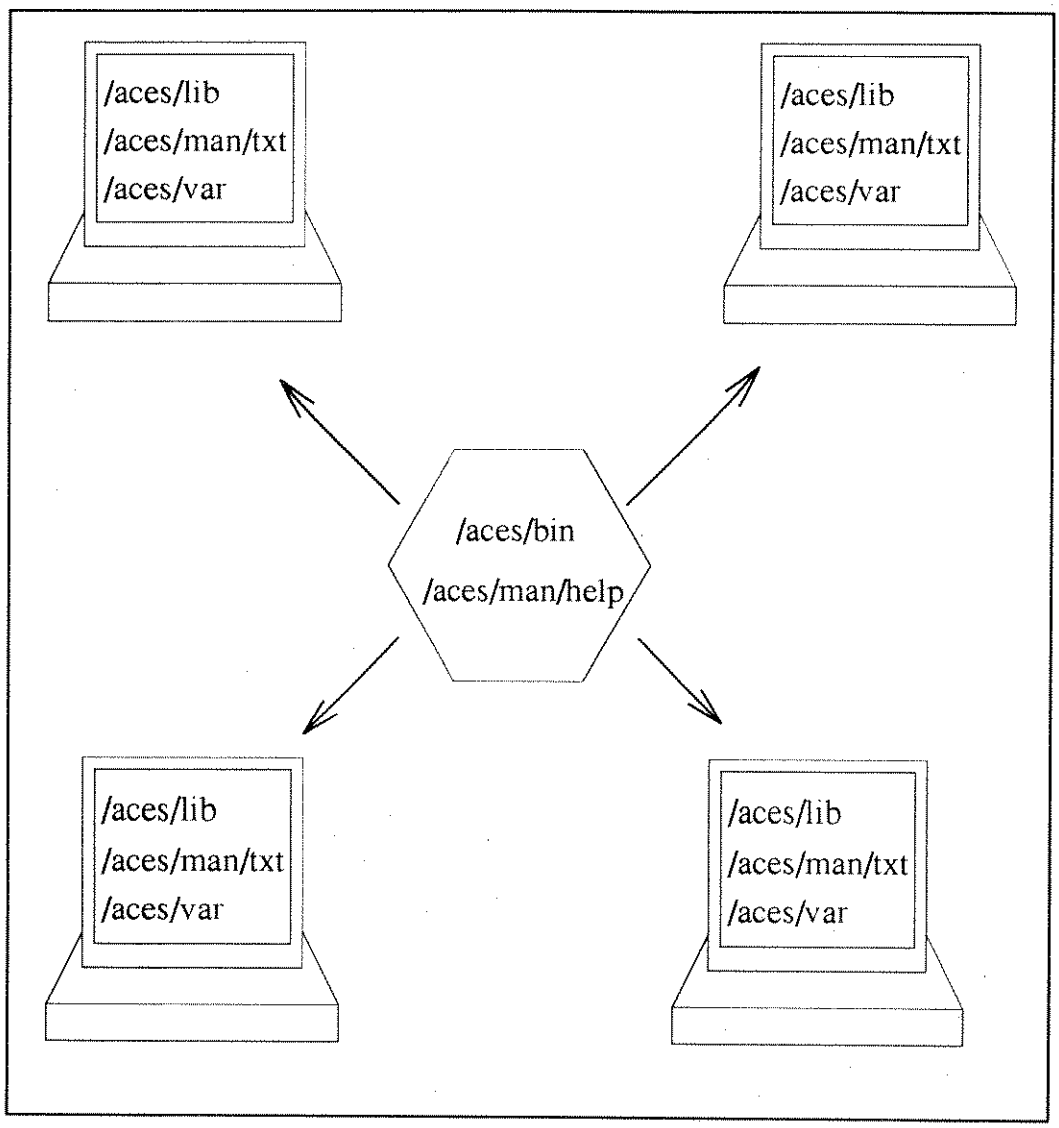

Figura 4.1: Esquema de distribuição dos diretórios em rede.

de métodos numéricos ou transformações. Estes programas são gerados externamente ao ambiente.

- Gráficos: As janelas que apresentam os gráficos possíveis de serem gerados para cada modelo se encontram pré-definidas. Elas são criadas por programas escritos usando funções do Xlib.
- Interfaces Gráficas: Neste grupo estão as janelas que exibem desenhos dos dispositivos físicos referenciados pelos arquivos de modelos. Além destas, há janelas que representam elementos de um painel de controle. Podemos citar, por exemplo, as janelas que imitam o *display*, analógico ou digital, de um instrumento de medida como o multímetro. São responsáveis pela interação propriamente dita entre usuários e modelos. Todas elas também são formadas por rotinas do Xlib e devem ser implementadas de modo a oferecer uma ampla utilização, englobando as características de um grande número de modelos.
- Textos: Todos os textos a serem acessados durante o uso deste ambiente estão reunidos num único local. Alguns deles fazem parte do guia de informações gerais sobre o programa, do manual de utilização e outros trazem as explicações sobre cada um dos modelos presentes no ambiente.

Dentro de cada um dos diretórios que contêm os elementos acima, pode haver uma classificação de acordo com a área de conhecimento a que eles se referem, mediante a criação de um subdiretório com o nome desta. Isto não impede, no entanto, de se utilizar um elemento de uma área em outra, uma vez que pode se empregar um mecanismo de referências (*links*), baseado na implementação para o sistema operacional UNIX.

O desafio da construção de um banco de dados, de modo a armazenar dados bastante heterogêneos como os apresentados acima e estabelecer o inter-relacionamento entre eles, está na busca de um modelamento adequado para a sua representação. Foi pensando nisso que definimos a *unidade* de estudo, chamada tópico.

O tópico de simulação resulta da união de alguns dos elementos representativos de cada um dos quatro segmentos do banco de dados. Ele compõe um assunto bem definido para simulação e segue, em sua formação, o princípio da herança e instanciação dos elementos, características da programação orientada a objetos.

Trata-se de um arquivo gerado a partir do editor do ambiente, contendo as declarações dos nomes do modelo, dos gráficos, dos textos e das janelas de interface gráfica, juntamente com chamadas de funções que, através da passagem de parâmetros, fazem a ligação entre estes elementos. Convém salientar a importância da utilização de uma norma para esta passagem de parâmetros. Estas funções formam uma linguagem própria para este banco de dados e são interpretadas por um programa que faz parte deste ambiente. A tradução é feita no momento em que se está usando o tópico.

Na figura 4.2 apresentamos em esquema a relação entre o tópico e seus componentes, enquanto que na figura 4.3 mostramos uma sugestão para a linguagem destinada à criação de tópicos. Como podemos observar nestes exemplos, o tópico representa uma nova visão para o banco de dados. Neste nível, o usuário que deseja montar um tópico, não necessita saber de que maneira foram implementados cada um dos seus elementos, mas apenas precisa das informações concernentes aos parâmetros envolvidos em um determinado modelo.

Diferentes usuários requerem visões distintas. Para o administrador do banco de dados, todos os detalhes de projeto devem estar visíveis, enquanto que para o usuário final, o programa é um conjunto de *caixas pretas*, importando somente a sua interface [3].

Os arquivos dos tópicos estão reunidos em diretórios que possuem como nome, por exemplo, as siglas das disciplinas de um determinado curso. Assim, todos os assuntos de uma disciplina que possam se transformar em elementos para simulação, devem ser reunidos em tópicos que serão armazenados no diretório desta disciplina.

O conjunto das disciplinas formam o que chamamos de *área do conhecimento*, que por sua vez também é um diretório e cujo nome é o da própria

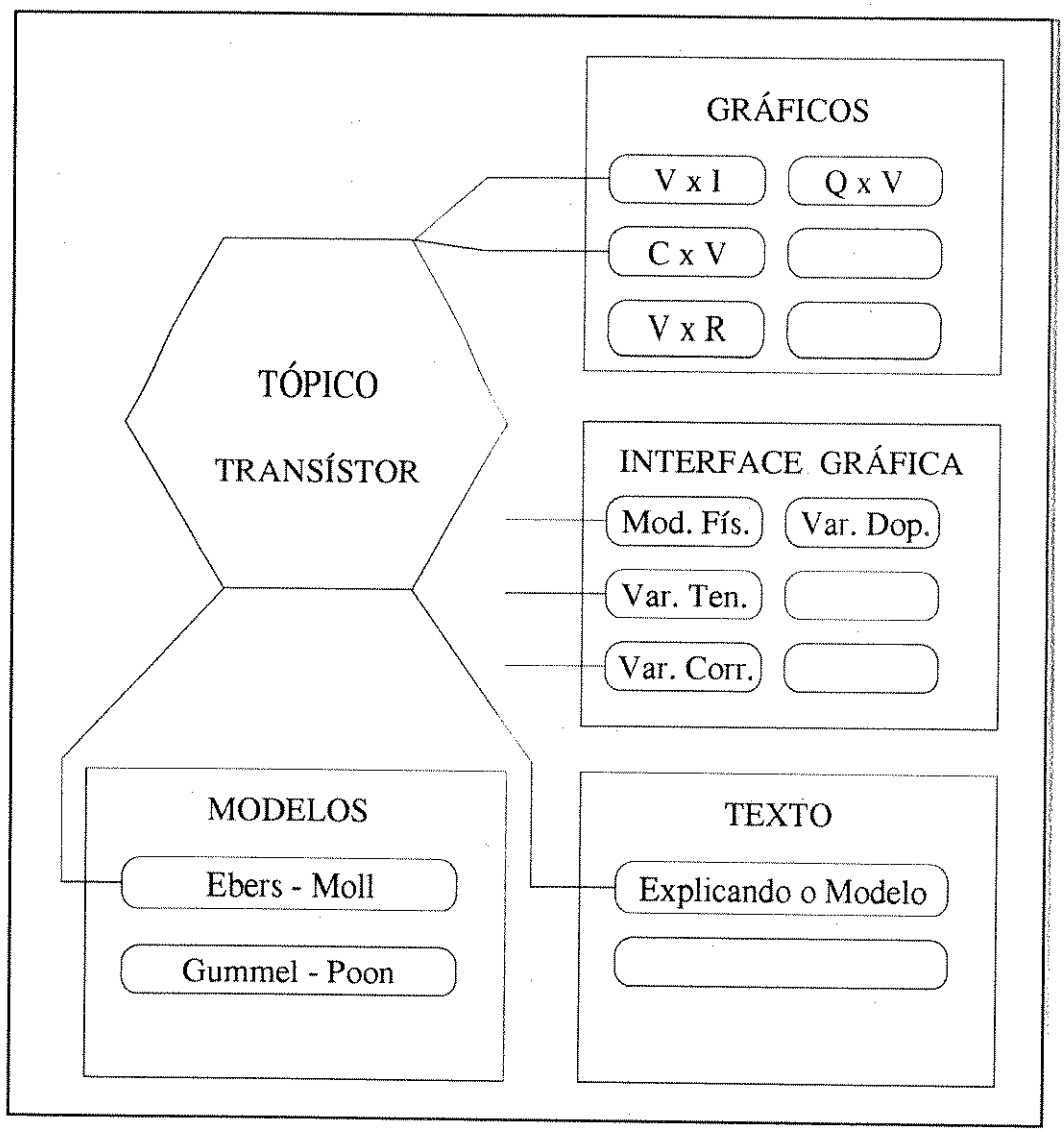

Figura 4.2: Relação entre um tópico e os elementos do banco de dados.

```
# Arquivo de Tópico
# Indicação do nome do tópico
  topic transistor
# Declarações do modelo, gráficos, subjanelas de interface e texto
  declare model ebers_moll
  declare graphs V_versus_I
  declare windows desenho_juncao, display_analog_corrente,
                  display_digital_tensao, controle_tensao
  declare text explicando_o_modelo
# Informação das condições iniciais para simulação
  temperature 300K
# Declaração das variáveis internas
  variables V, I
# Programa principal
  beginloop
    get(ajuste_tensao, V)                 # obtém o valor da tensão V
    put(display_digital_tensao, V)        # apresenta o valor de V na subjanela
    calculate(ebers_moll, V, I)           # calcula a corrente I
    put(display_analog_corrente, I)       # apresenta o valor de I na subjanela
    put(V_versus_I, V, I)                 # apresenta o gráfico VxI
  endloop
```

Figura 4.3: Exemplo de uma linguagem para a formação de tópicos.

área.

Numa última etapa, estas áreas encontram-se em um diretório para o qual sugerimos o nome **/aces/areas**. Assim, esta múltipla estrutura de diretórios passa também a fazer parte do ambiente. Como já foi discutido anteriormente, esta estrutura para as áreas, disciplinas e tópicos pode estar distribuída pela rede de estações de trabalho. Dessa maneira, obtemos a independência entre as áreas juntamente com o compartilhamento das informações. A figura 4.4 apresenta o esquema aqui detalhado.

Com relação à manutenção do banco de dados em geral, observamos que isto deve ser feito por alguém com uma certa experiência em linguagens de programação e em banco de dados, além de estar familiarizado com a estrutura deste ambiente.

4.6 O Ambiente do Ponto de Vista do Usuário

O ambiente desenvolvido propõe um mecanismo de controle de acesso em relação aos seus usuários, de maneira a regulamentar o compartilhamento de recursos e monitorar o seu uso. O usuário poderá utilizar este programa desde que seu *login name* esteja incluso num arquivo apropriado de cadastro de usuários e deverá estar inscrito em uma das seguintes categorias:

- Alunos:

  Têm permissão para acionar os programas de simulação, ativar programas externos, gerar arquivos para impressão ou colocá-los na sua área em disco e utilizar o sistema de comunicação interno.

- Professores:

  Além de possuírem as mesmas permissões dos alunos, podem inserir novos módulos de simulação neste ambiente, criados através do editor.

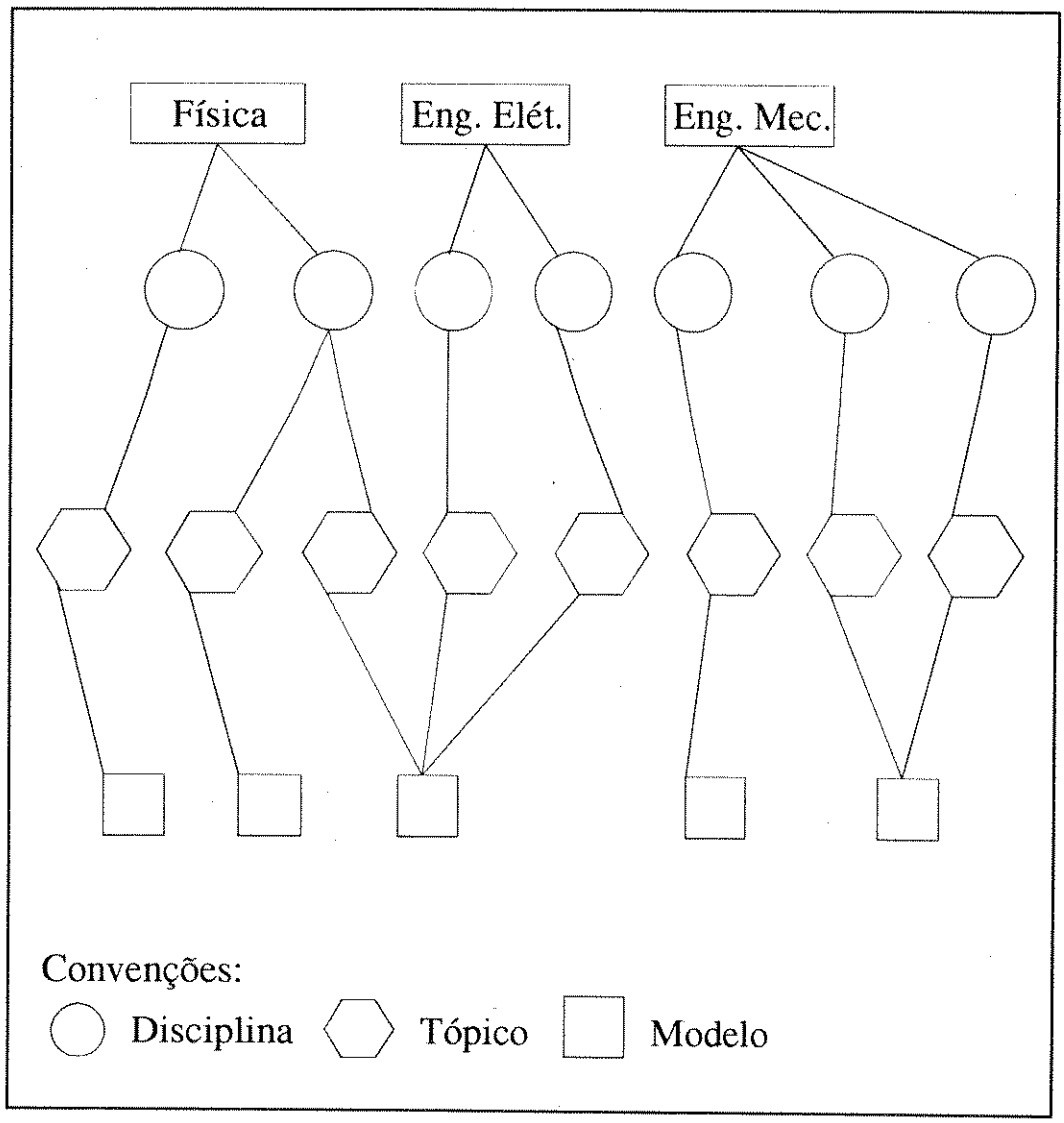

Figura 4.4: Estrutura geral dos diretórios das áreas de conhecimento.

- Super-Usuário:

  Tem as mesmas permissões dos demais usuários e é o único com privilégio de acesso a todos os arquivos do ambiente, incluindo o banco de dados.

No caso em que um usuário não esteja encaixado em uma destas categorias, ele terá como único recurso disponível uma opção de demonstração. Será apresentada uma janela de simulação completa, através da qual o usuário poderá conhecer uma amostra dos recursos oferecidos pelo ambiente.

Toda a parte de ativação de programas ou botões é sensível à categoria, ou seja, uma vez encontrado o nome do usuário no arquivo de cadastro, é mediante a verificação de sua categoria que o programa irá lhe permitir ou não a execução de diversas tarefas. Portanto, isto dispensa a necessidade da implantação de qualquer esquema de senhas para os usuários.

Com relação ao super-usuário observamos a importância de haver um para cada área, na qual teria privilégios totais. Isto evita com que um super-usuário venha a alterar, por exemplo, arquivos do banco de dados ou referências a estes (*links*) de uma área que não seja a sua.

4.7 Elementos da Interface com o Usuário

A interface com o usuário é feita através da utilização de um conjunto de janelas. Elas estão constiuídas, de maneira geral, por uma área de controle e de apresentação.

A área de controle, neste ambiente, é formada por botões, menus, campos para digitação de texto (*text fields*), ítens de opções de assinalamento exclusivo (*exclusive settings*) e ítens de opção tipo liga-desliga (*check boxes*). Esta área é responsável através dos seus elementos, pela entrada de comandos do usuário em relação ao programa [10, 11, 12].

A área de apresentação, que mostra o resultados dos comandos ao usuário, tem como elementos principais neste programa, as subjanelas que pos-

suem um controle utilizado para a movimentação vertical ou horizontal de uma lista de informações apresentadas, chamado *scrollbar*.

Cada um dos elementos representantes das áreas acima possui características próprias, a saber [10, 11, 12]:

- Botões: Constituem o meio mais comum para a entrada de comandos. Possuem geralmente forma retangular, sendo que no seu interior encontra-se o nome do comando que é executado após o seu acionamento.
- Menus: Ativados a partir de um botão e são formados por uma área retangular contendo uma lista de opções para escolha.
- *Text fields*: São campos reservados à entrada de dados via teclado. O tamanho do texto não se limita às dimensões do campo, uma vez que há mecanismos de *deslocamento* das informações digitadas, com indicação para o usuário através do posicionamento de uma barra vertical à direita ou à esquerda do campo.
- *Exclusive settings*: Ítens posicionados de maneira horizontal ou vertical, cada qual apresentando o nome de uma opção ao lado de um pequeno quadrado onde é assinalada a sua seleção. A escolha é feita de maneira exclusiva, isto é, somente um item dentro da lista pode ser selecionado a cada vez.
- *Check box*: Possui a mesma apresentação dos *exclusive settings*. No entanto, caracteriza-se como um item isolado que configura apenas dois estados, selecionado ou não.
- *Scrollbar*: É um elemento ligado à uma subjanela, cuja finalidade é tornar possível a apresentação de uma grande quantidade de informações em um espaço restrito. Através dele o usuário pode fazer o deslocamento vertical ou horizontal do texto inscrito em uma subjanela.

Outro importante recurso existente nesta interface é a possibilidade de configurar cada um dos ítens acima como ativo ou inativo (*dimmed*). Neste caso, ele não aceita a entrada de ações vindas do teclado ou *mouse*.

A figura 4.5 exemplifica cada um dos elementos discutidos nesta seção.

A implementação destes ítens foi feita através da adaptação de algumas funções, escritas utilizando rotinas do Xlib, provenientes do programa XV que faz captura e apresentação de imagens.

Estas funções se encontram em arquivos distintos, agrupadas de acordo com o elemento que representam. O conjunto destes arquivos forma o *toolkit* da interface gráfica deste ambiente.

Com a indicação de alguns parâmetros característicos de cada item da interface como, por exemplo, coordenadas de posicionamento na janela, dimensões, cor e nome, pode-se fazer uso das funções deste *toolkit* em qualquer programa, sem haver a necessidade de conhecimento do seu código em linguagem de programação.

4.8 Hipertexto

Hipertexto é uma maneira moderna, rápida e eficiente de se obter informações a respeito de um assunto, usado freqüentemente em ambientes computacionais.

Ele difere da maneira tradicional de apresentação de textos em razão da sua mobilidade. O texto é apresentado na tela do computador havendo algumas palavras que se encontram em destaque, seja pela diferença no padrão, na cor ou ainda pela presença de um linha de contorno, às quais chamamos de *palavras-chave*.

Através do *mouse* o usuário pode selecionar a palavra-chave desejada e como conseqüência será apresentada a ele uma nova página de texto que contém as informações referentes à palavra escolhida.

Trata-se, portanto, de um mecanismo de consulta dirigida, no qual o usuário não precisa percorrer todo o documento para encontrar uma informação, mas pode escolher somente os trechos correspondentes àquelas palavras que são

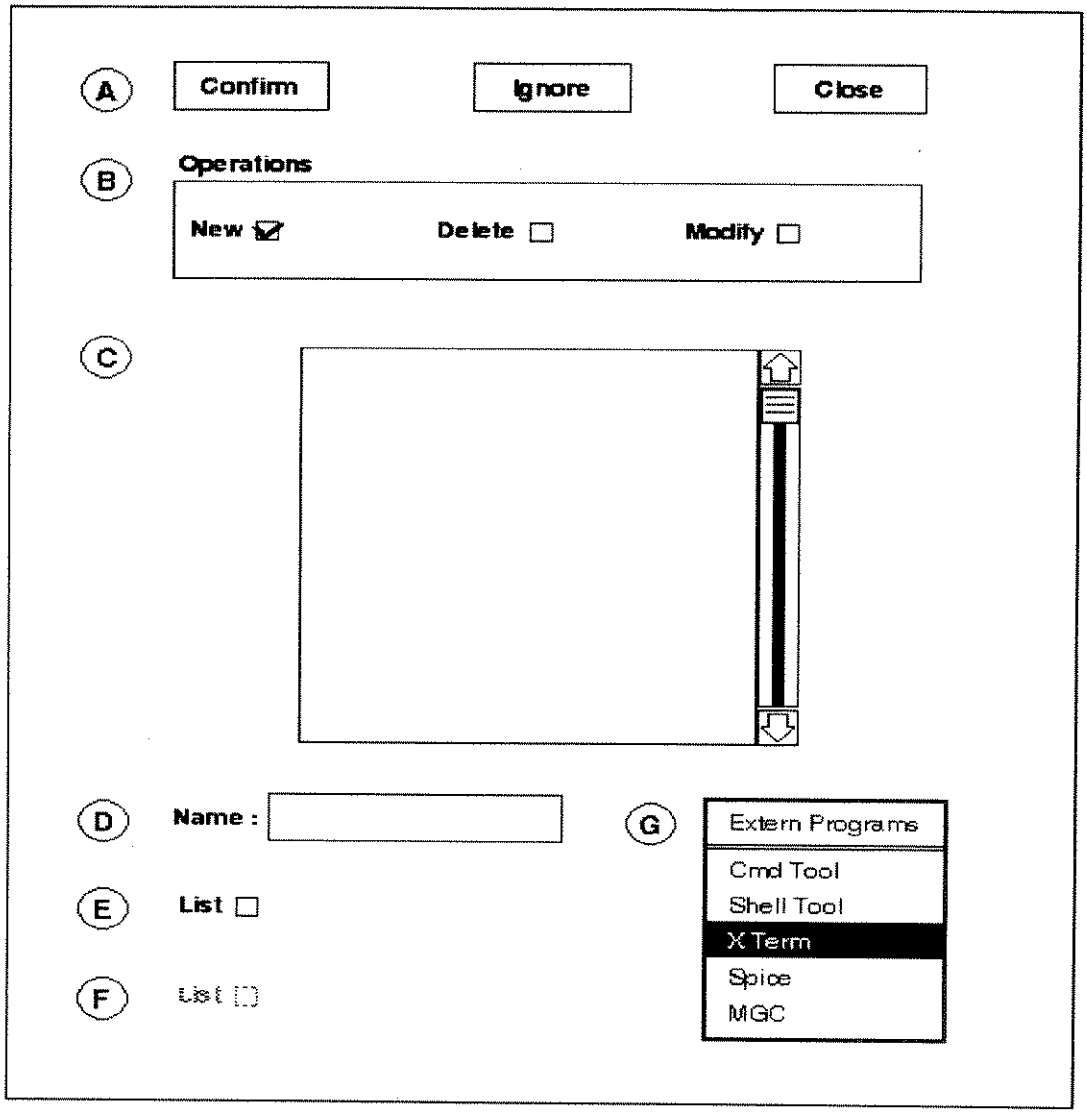

Figura 4.5: Elementos da interface com o usuário: a) Botões b) *Exclusive settings* c) Janela com *Scrollbar* d) *Text field* e) *Check box* (ativo) f) *Check box* (inativo) g) Menu.

importantes a ele no momento.

Em nosso ambiente prevemos a existência de um esquema de hipertexto que vai atuar em todos os níveis de informação, seja com relação aos manuais do ambiente ou aos textos referentes aos modelos.

Este esquema é baseado nos seguintes componentes: uma série de arquivos de textos armazenados em um diretório próprio dentro do ambiente e um arquivo de configuração que contém os dados para localização de um texto a partir de uma palavra-chave.

Este arquivo de configuração é formado por linhas que contém os seguintes campos:

- Palavra-chave: É a palavra que representa a ligação para uma outra página de texto.
- Arquivo origem: Nome do arquivo onde a palavra-chave se encontra em destaque. Este campo é importante pois permite que uma mesma palavra, dependendo da sua localização, tenha um texto diferente a ela relacionado. Isto significa que a referência entre as palavras-chave e os trechos de um documento é contextual.
- Arquivo destino: Nome do arquivo onde estão as novas páginas de texto a serem apresentadas ao usuário.

Para o bom funcionamento do mecanismo de hipertexto necessitamos de um programa dedicado. É ele o responsável pela busca dos textos na estrutura de diretórios do ambiente, tendo como referência as informações presentes no arquivo de configuração.

Este programa se encarrega também da formatação das páginas de um arquivo de documento. Ele faz o ajuste do número de linhas e colunas em cada página, sendo que as informações para esta formatação se encontram no próprio arquivo.

No momento da leitura do texto, o programa pode manter algumas páginas do documento em memória, de maneira a não sobrecarregar o acesso ao

disco. Também pode armazenar temporariamente, o caminho na estrutura de diretórios (*path*) de cada arquivo de documento já acessado, de modo a ganhar velocidade na recuperação das informações.

A figura 4.6 exemplifica a estrutura do hipertexto.

4.9 Estatística

A estatística tem como objetivo apresentar informações aos professores e aos super-usuários, para que possam acompanhar a utilização, respectivamente, dos tópicos de simulação por parte dos alunos e do ambiente como um todo. Está dividida em duas formas:

- Quantitativa: Traz informações com relação ao tempo de utilização por área de conhecimento, disciplina e tópico. Também fornece informações relativas ao tempo gasto por aluno durante as sessões de simulação.

  Uma observação importante diz respeito ao período de tempo para a amostragem estatística. A cada dia o programa se encarrega de gravar as informações num arquivo distinto. Após 30 dias é criado um novo arquivo com os dados de todos os arquivos diários, sendo estes automaticamente apagados. O processo se repete e ao final de um ano, nova coleção de dados é feita, baseada nos arquivos mensais. Este esquema é empregado no mecanismo de registro de uso do sistema operacional UNIX, denominado *account* [14].

  O processo acima descrito é uma sugestão, podendo ser alterado caso os arquivos gerados representem um gasto excessivo de memória em disco.

- Qualitativa: Relaciona as áreas, disciplinas e tópicos acessados pelos usuários. Estas informações são gravadas de forma seqüencial, permitindo a verificação da ordem de exploração do ambiente por parte de cada usuário.

  No que se refere ao armazenamento destes dados em arquivos, sugerimos a adoção de um esquema semelhante ao proposto para as informações

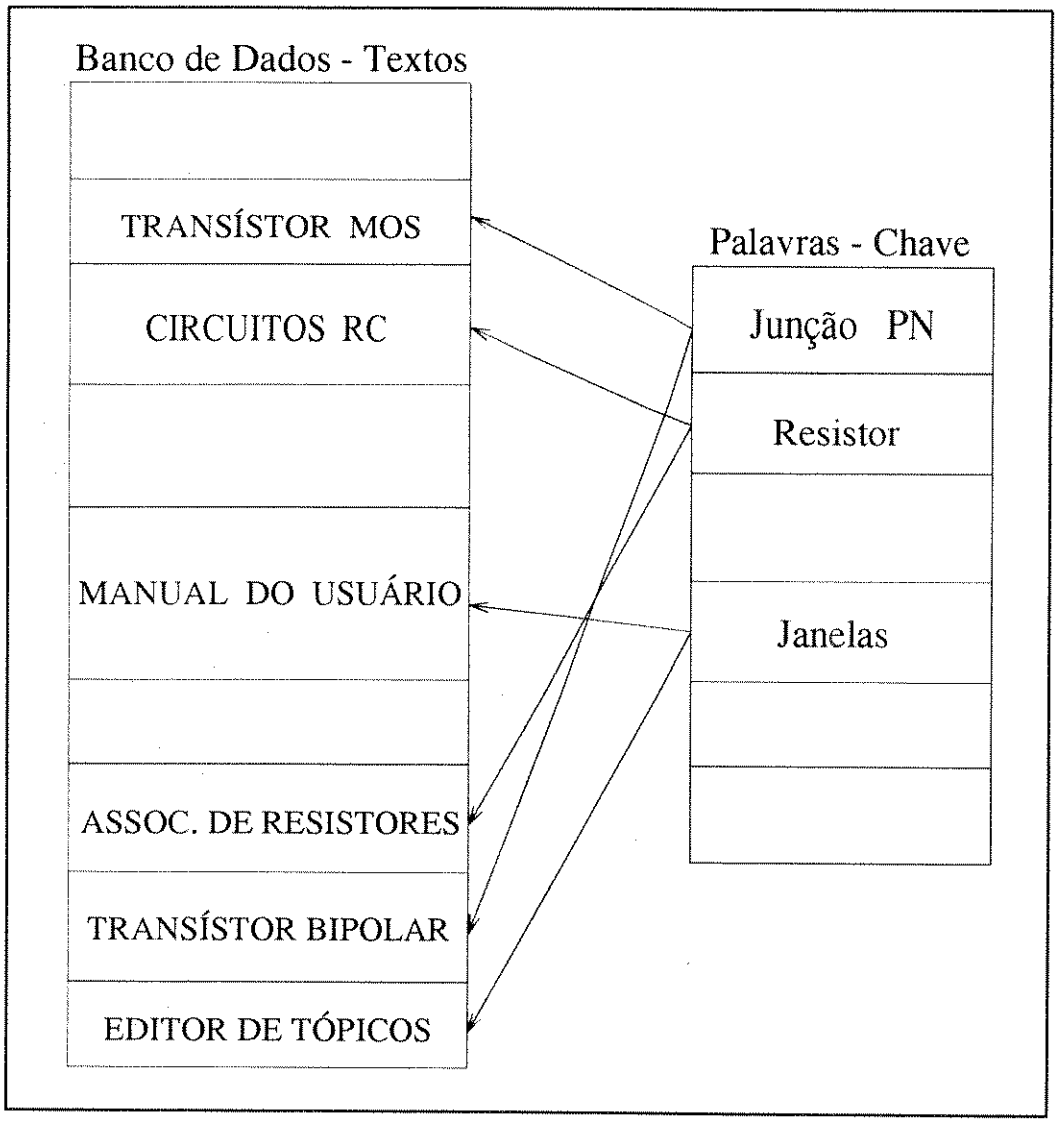

Figura 4.6: Esquema da estrutura do hipertexto.

quantitativas, excetuando-se a necessidade de fazer o resumo anual. Assim, ao final de um período de 30 dias, todos os arquivos relativos à estatística qualitativa seriam apagados.

4.10 Arquivos de Controle e de Mensagens

Através destes arquivos o gerenciador pode executar todas as tarefas descritas para este ambiente.

Formam o grupo de controle do ambiente os arquivos: de usuários, das áreas e disciplinas, dos dados estatísticos, de referência para o hipertexto e de sessão.

O arquivo de usuários é consultado de modo a verificar a permissão de acesso ao ambiente e às suas opções. Possui a seguinte estrutura:

- Nome: Contém o *login name* do usuário, ou seja, o nome como ele é conhecido pelo sistema operacional UNIX.
- Áreas: Relação dos identificadores de cada área de conhecimento.
- Categoria: É um número que especifica se o usuário é aluno, professor ou super-usuário.
- Disciplinas: Uma lista das siglas das disciplinas que podem ser acessadas por alunos e professores. No caso do super-usuário este campo está em branco.

O arquivo das áreas e disciplinas é utilizado quando se deseja obter os seus nomes, bem como a relação de pertinência entre disciplinas e áreas. Tem como estrutura:

- Um número que identifica uma área internamente ao ambiente.
- Nome da área de conhecimento.
- Sigla e nome completo de cada disciplina.

Os arquivos de sessão estão localizados na *área de login* de cada usuário. Sua função é armazenar as informações com relação à configuração das janelas do ambiente quando do último acesso por parte do usuário. Isto possibilita ter numa próxima ativação do ambiente, a apresentação imediata de todas estas janelas, abreviando o trabalho do usuário. É constituído por:

- Nome da janela interno ao programa.
- Posição da janela em relação à tela da estação de trabalho.

Os arquivos de mensagens são provenientes do sistema de correio eletrônico exclusivo. Eles não apresentam limite quanto ao número de mensagens que possam conter. Cada usuário possui um destes arquivos, identificados pelo seu *login name*, a exemplo do que ocorre no sistema operacional UNIX [14]. A estrutura de uma mensagem é a seguinte:

- Usuário: *Login name* do usuário que enviou a mensagem.
- Data: Informações relativas à data e hora de chegada da mensagem.
- Assunto: Algumas palavras que resumem o conteúdo da correspondência.
- *Status*: Uma letra indicando se a mensagem foi lida ou não.
- As linhas de texto que formam a mensagem.

4.11 Gráficos

Os gráficos, de maneira geral, representam um meio rápido e seguro para a obtenção de informações.

Neste ambiente eles são bastante importantes pois vêm mostrar, em primeiro lugar, os resultados das simulações dos modelos. Além disso, podem ser gerados a partir de arquivos de coordenadas fornecidos pelo usuário ou por outro programa.

Haverá a possibilidade da criação de gráficos estáticos, que apresentam um resultado definido para um conjunto de valores, ou dinâmicos, que exibem a curva de pontos levando em consideração a variação de parâmetros que ocorre a cada instante. Eles também poderão ser bi ou tridimensionais e o usuário terá condição de gravá-los na sua *área de login*.

A impressão dos gráficos pode ser feita através da conversão de seus pontos em um formato padrão, como por exemplo, *bitmap* ou *PostScript*.

4.12 Sistema Especialista

O conceito geral de um sistema especialista é o de um programa que procura simular o raciocínio de um especialista humano em algum domínio do conhecimento.

É composto basicamente, de uma coleção de informações sobre um determinado assunto e de um conjunto de regras que fazem o seu relacionamento. Sua função vai além de responder às consultas fornecidas pelo usuário. Pode também diagnosticar problemas e em alguns casos, até demonstrar o *raciocínio* para a obtenção de uma conclusão.

Uma característica importante do sistema especialista é que o seu banco de dados seja modificável, de modo a oferecer ao usuário informações sempre atualizadas.

Neste ambiente prevemos a introdução de um sistema especialista com a finalidade de responder às perguntas dos alunos com relação aos modelos usados nas simulações. Além disso, poderá vir a analisar dúvidas colocadas através do sistema interno de correio eletrônico.

Ele deverá ser implementado em uma linguagem dedicada para a criação de sistemas especialistas e utilizará um banco de dados exclusivo, podendo porém em determinados momentos, interagir com aquele do própio ambiente.

Capítulo 5

Sistema de Janelas e Opções

Em cada seção deste capítulo apresentamos a funcionalidade das principais janelas que compõem a interface gráfica do ambiente. Descrevemos cada uma das opções existentes e a forma com a qual os resultados são exibidos.

Convém salientar que todas as opções se encontram escritas em inglês. Esta escolha foi motivada em razão de sua praticidade na definição de comandos e porque a consideramos língua *padrão* para uso em programas computacionais.

5.1 Janela Inicial

O acionamento do programa através de uma estação de trabalho, trará à tela uma janela com um conjunto de botões de controle. Figura 5.1.

Na metade superior desta janela encontramos os botões responsáveis pela ativação de uma série de operações executadas pelo ambiente. Complementando-a, estão presentes os botões relacionados às áreas de conhecimento para simulação. Como exemplo, escolhemos as áreas de Engenharia Elétrica, Engenharia Mecânica, Engenharia Química, Física e Química.

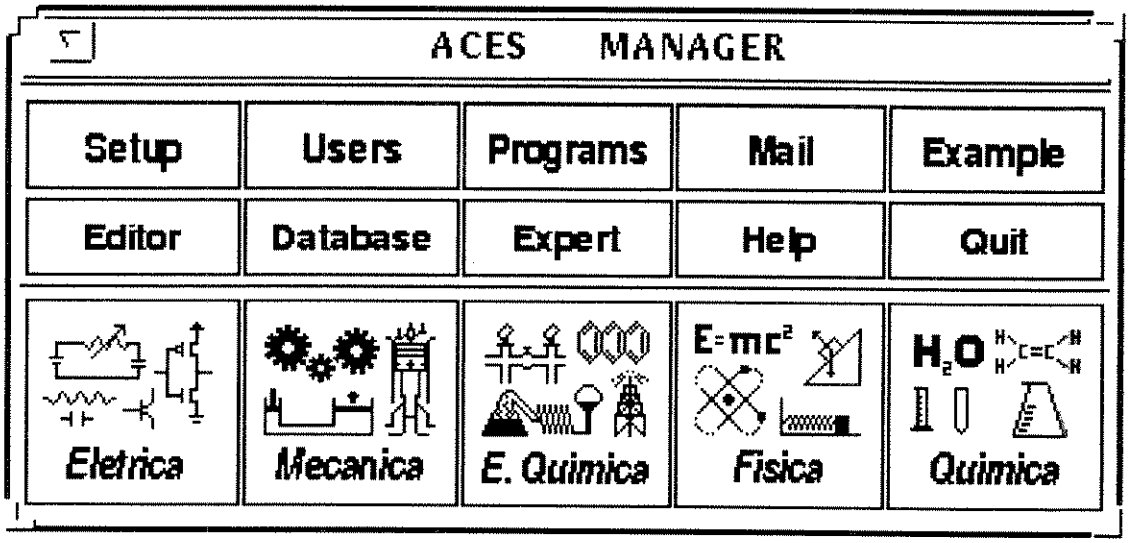

Figura 5.1: Janela de entrada do ambiente.

5.2 Setup

Através da seleção do botão **Setup**, é apresentado ao usuário um menu com três opções.

A primeira permite a recuperação imediata da configuração das janelas da última sessão de uso do ambiente. As informações necessárias para esta recuperação estão armazenadas em um arquivo que se localiza no diretório principal da *área de login* de cada usuário.

Na escolha da segunda opção o usuário obtém a lista das últimas disciplinas acessadas. Fixamos em oito, como sugestão, o número de disciplinas presentes nesta lista, que por sua vez também se encontra em um arquivo dentro da *área de login* do usuário.

A última opção deste menu apresenta a lista de disciplinas, dentro de cada área de conhecimento, para as quais o usuário tem acesso para fazer as simulações. Esta lista se encontra no arquivo de usuários que verifica a permissão de acesso ao ambiente.

As informações fornecidas por estas duas últimas opções são apresentadas em janelas independentes, constando a sigla e o nome completo das disciplinas.

5.3 Users

O acionamento do botão **Users** traz à tela da estação uma nova janela que é responsável pelas operações de cadastramento de usuários, apresentação dos dados estatísticos e de opções de consulta aos usuários e disciplinas. Figura 5.2.

Na metade superior desta janela temos a parte destinada ao controle de usuários. Em primeiro lugar aparecem as operações que podem ser feitas no arquivo de usuários, destacadas pela palavra **Operations**. Estas operações compreendem a inclusão (**New**), exclusão (**Delete**) e modificação (**Modify**) de dados do usuário.

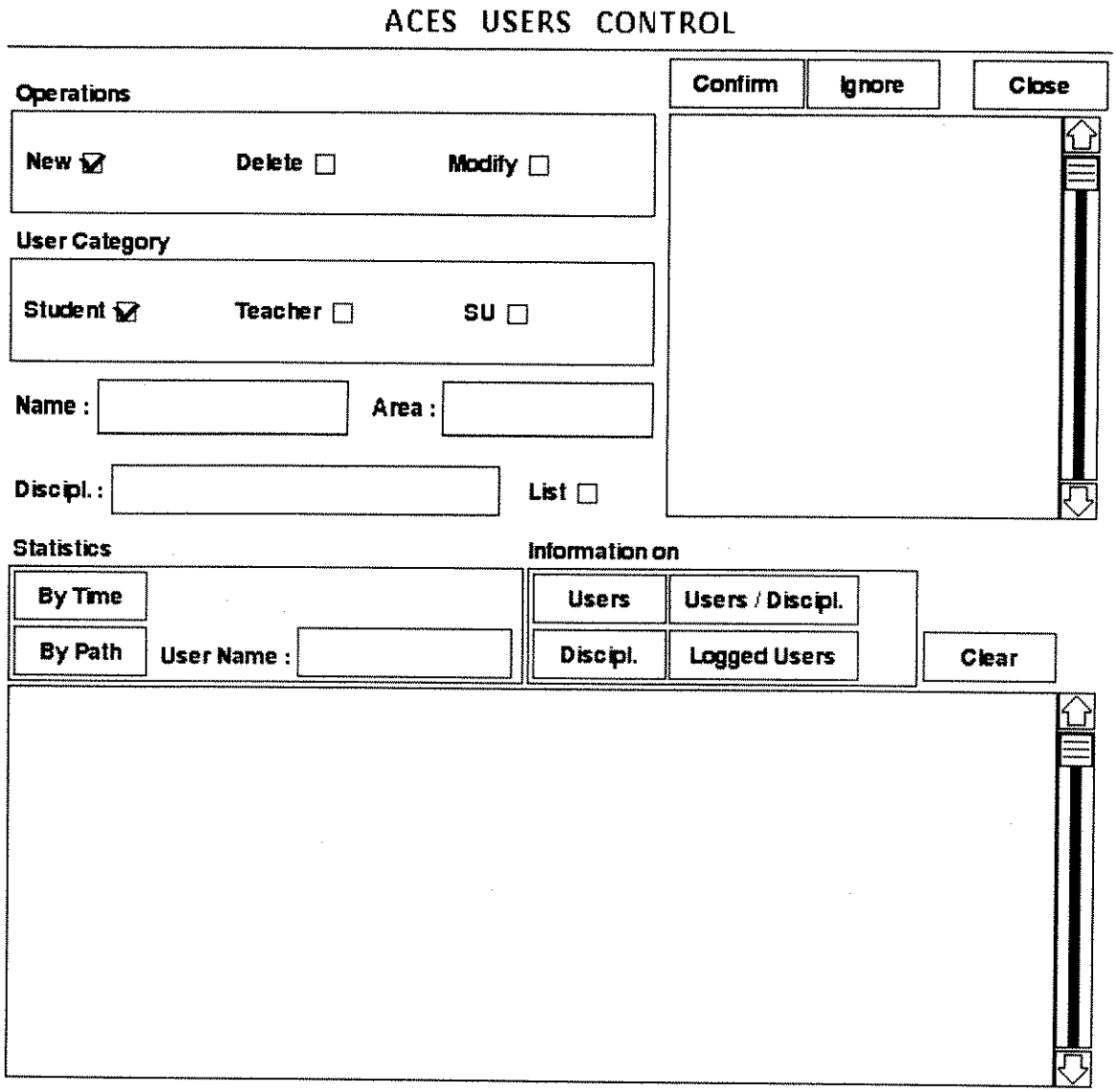

Figura 5.2: Janela apresentada pela opção **Users**.

Logo abaixo, em **User Category**, são apresentadas as opções para escolha da categoria do usuário. Aluno, professor e super-usuário estão representados, respectivamente, por **Student**, **Teacher** e **SU**.

Seguem-se os espaços reservados para a digitação do *login name* do usuário, da área de conhecimento de sua atuação e da lista de disciplinas para as quais terá acesso.

De maneira a tornar mais rápida a recuperação das informações sobre os usuários e agilizar o processo de cadastramento, foi colocada a opção **List**, que é reponsável quando selecionada, pela apresentação de uma listagem dos usuários, áreas ou disciplinas, na subjanela que se encontra ao lado. Esta listagem é sensível ao contexto, isto é, no caso da apresentação dos nomes dos usuários, ela se realiza de acordo com a categoria escolhida. Da mesma maneira se processa a listagem das disciplinas, levando em conta a área à qual pertencem.

A efetivação das operações descritas acima, ou seja, a gravação em arquivo, está condicionada a uma confirmação ou não, que é fornecida pelos botões **Confirm** e **Ignore**, respectivamente.

Na metade inferior da janela **Users** temos as opções que apresentam as informações estatísticas (**Statistics**) e de consulta aos usuários e disciplinas (**Information on**).

Os dados estatísticos de utilização do ambiente podem ser mostrados pela ativação dos botões **By Time** ou **By Path**. O primeiro apresenta um menu com quatro opções de escolha. Estas opções trazem as informações com relação ao tempo gasto na utilização, agrupadas por disciplina, tópico ou aluno e também de forma global. O segundo relaciona os nomes dos tópicos, disciplinas e áreas aos quais o usuário tenha feito acesso. Para isso, antes de pressionar esse botão, deve-se digitar o *login name* do usuário no campo reservado, ao lado.

As consultas aos usuários e disciplinas são realizadas através dos botões **Users**, que apresenta a listagem de todos os usuários do ambiente e **Discipl.**, que oferece a listagem das disciplinas em suas respectivas áreas. Além destes, temos o botão **Users/Discipl.** que mostra a lista dos usuários em cada

disciplina e o botão **Logged Users** que faz a listagem dos usuários que estejam utilizando o ambiente em determinado momento.

Todas as informações estatísticas e de consulta aos usuários e disciplinas são apresentadas na subjanela que se encontra abaixo dos botões que executam estas opções. O botão **Clear** realiza a tarefa de *limpar* esta subjanela, entre as consultas.

As opções existentes na janela **Users** são sensíveis à categoria do usuário. Isto significa dizer que, se um aluno estiver utilizando o ambiente, nesta janela só estarão ativas as opções de consulta aos usuários e disciplinas. No caso de um professor, somente estará inativa a opção de categoria **SU**, posto que apenas um super-usuário pode fazer o cadastramento de outro e indicar a área pela qual este passa a ser responsável.

Finalmente, o botão **Close** executa o fechamento desta janela.

5.4 Programs

Através desta opção tem-se a possibilidade de interação com outras aplicações disponíveis na estação de trabalho. É apresentado ao usuário um menu, no qual estão os nomes de alguns programas externos ao ambiente e que podem ser executados de acordo com a sua escolha.

Após o acionamento de qualquer utilitário ou programa externo, este passa a ser totalmente independente do ambiente de simulação.

5.5 Mail

Quando da ativação da opção **Mail**, é apresentada uma janela destinada à leitura, composição e envio de mensagens. Figura 5.3.

A subjanela superior está reservada à apresentação do cabeçalho das mensagens destinadas ao usuário. Este cabeçalho é formado pelo número da

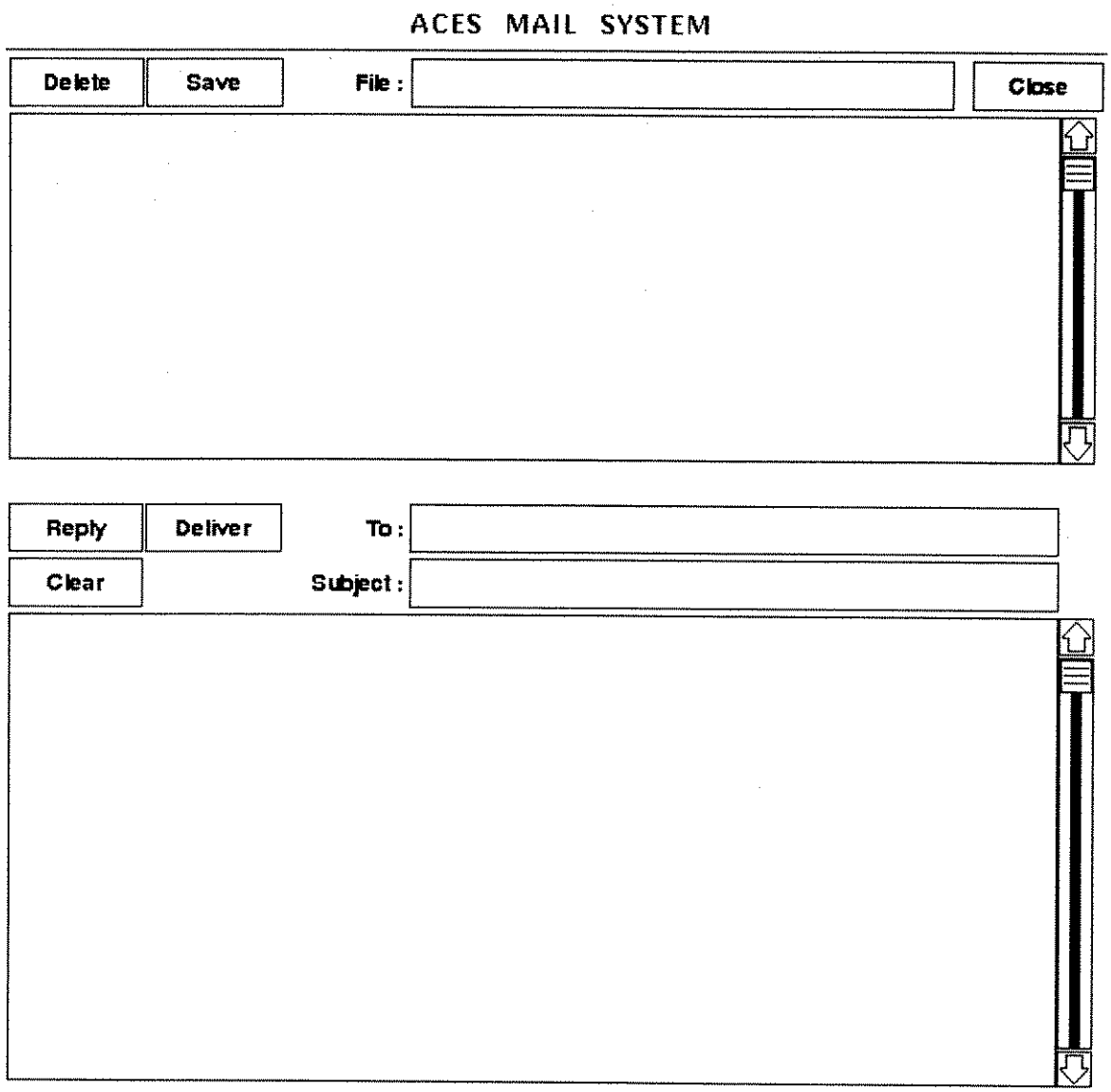

Figura 5.3: Janela de opções do sistema de correio eletrônico.

mensagem, nome do remetente, data e hora de chegada e assunto. Para ler uma mensagem, basta o usuário levar o *mouse* à linha de cabeçalho desejada e pressionar duas vezes o seu botão de seleção. Assim, uma nova subjanela com o conteúdo da mensagem irá aparecer. Na escolha de uma outra mensagem para apresentação, o conteúdo da anterior será retirado desta subjanela.

Para guardar as mensagens em um arquivo, o usuário deve digitar o nome deste no campo indicado por **File**, selecionar as mensagens e pressionar o botão **Save**. A eliminação de uma mensagem previamente selecionada é feita com a ativação do botão **Delete**.

A composição de mensagens é realizada na subjanela inferior. Acima dela temos um campo reservado à colocação do *login name* do destinatário (**To**) e outro para a indicação do assunto **Subject**. Esta subjanela e os campos são *limpos* automaticamente após o envio de uma mensagem, que é a tarefa botão do **Deliver**.

Havendo uma mensagem selecionada na subjanela de apresentação e ao ser pressionado o botão **Reply**, os campos **To** e **Subject** já são preenchidos com as informações que constam no cabeçalho. Isto facilita a troca de mensagens entre usuários quando se está tratando de um mesmo assunto.

O botão **Clear** é responsável pela *limpeza* da subjanela de edição de mensagens, enquanto que o botão **Close** faz o fechamento da janela de **Mail**.

5.6 Example

Pressionando-se o botão **Example**, o programa faz a apresentação de uma janela de demonstração. Ela é constituída de um conjunto de subjanelas que representam uma interface completa de um tópico de simulação. A figura 5.9 mostra um esquema desta janela.

Através dessa opção, o usuário pode ter uma idéia melhor dos recursos existentes para a representação e interação com os modelos.

5.7 Editor

Através do botão **Editor**, professores e super-usuários têm acesso à janela do editor de tópicos. Figura 5.4.

Trata-se de uma ferramenta que permite a criação de um tópico a partir dos elementos presentes no banco de dados, tendo por base uma linguagem própria, com características de programação orientada a objetos. Dessa maneira, o usuário não precisa ter experiência como programador para montar um tópico, pois é necessário apenas fazer referências, através de alguns comandos, aos elementos desejados.

Observando a janela do editor temos, do lado esquerdo, um grande retângulo destacado pelas palavras **Topic Creation** que concentra os campos destinados, na sua maioria, à digitação dos nomes do tópico e dos seus elementos formadores (**Name**) e dos nomes dos arquivos em que estes se encontram (**File**).

O nome de um elemento é uma referência interna ao ambiente, para efeitos de programação, enquanto que o nome do arquivo é tratado pelo sistema operacional. Nos campos superiores são digitados estes nomes para o tópico e logo abaixo, para o modelo, no retângulo em que aparece a palavra **Model**.

Na região intitulada **Window**, inserem-se os nomes para as subjanelas da interface gráfica juntamente com uma seqüência de variáveis (**Var.**), que fazem parte do modelo. Elas terão seus valores apresentados nestas subjanelas durante a fase de simulação. Este procedimento é o mesmo para os campos contidos na região chamada **Graphic**, que diz respeito às subjanelas de gráficos.

Os campos relacionados aos arquivos de texto encontram-se no espaço indicado pela palavra **Text**. Além dos nomes, conforme explicado acima, pode-se digitar no campo **Key Words**, uma série de palavras-chave que serão incorporadas ao sistema de hipertexto. O botão **Delete** é responsável pela eliminação de palavras-chave, previamente escolhidas para o texto que esteja sendo digitado.

A exemplo do que foi descrito na seção relativa à janela **Users**, a opção **List** fornece a listagem dos nomes dos arquivos presentes no banco de

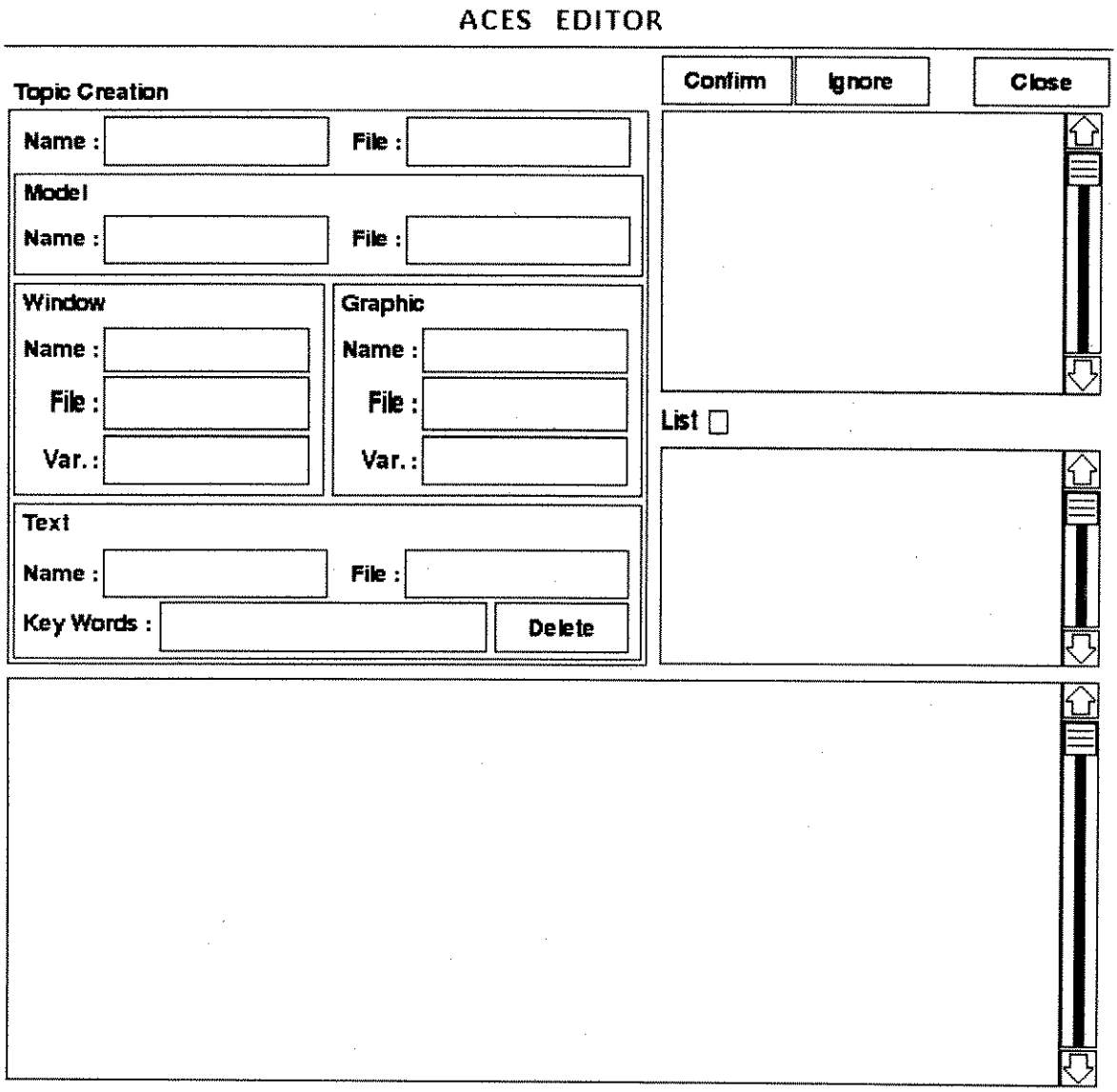

Figura 5.4: Janela para utilização do editor de tópicos.

dados e das palavras-chave existentes em todo o arquivo de configuração do hipertexto, incluindo ainda o número de referências a cada uma delas. Estas listagens são apresentadas, respectivamente, na subjanela acima e abaixo desta opção.

Os comandos da linguagem para a criação de um tópico e o texto de explicação de um modelo são digitados na subjanela inferior. Com relação a este último, são incluídos também comandos para a sua formatação, que posteriormente serão interpretados pelo sistema de hipertexto.

Os botões **Confirm** e **Ignore** respondem, respectivamente, pela confirmação e descarte dos procedimentos realizados para a criação de um tópico. O botão **Close** faz o fechamento desta janela.

5.8 Data Base

A opção **Data Base**, restrita ao super-usuário, exibe uma nova janela, através da qual ele poderá realizar operações no banco de dados do ambiente. A figura 5.5 apresenta esta janela.

No retângulo denominado **Operations** estão relacionadas as opções de inclusão (**Include**), exclusão (**Delete**), modificação (**Modify**) e consulta (**Consult**) de qualquer elemento pertencente ao banco de dados, que pode ser selecionado dentre as opções existentes na região destacada por **Elements**. Esta, por sua vez, se encontra dividida em duas partes.

Na primeira fazemos a escolha do modelo (**Model**), da subjanela da interface gráfica (**Window**), do texto (**Text**) ou da subjanela para gráficos (**Graphic**). Há um campo para digitação do nome do arquivo correspondente (**File**).

Na outra parte, escolhemos a disciplina (**Discipl.**), o tópico (**Topic**) ou a referência a esses elementos (**Link**), inserindo-se neste caso, o local de origem (**From**) e de destino (**To**). Com relação à disciplina, deve-se digitar o seu nome completo e a sua sigla nos campos indicados por **Name** e **Cod.**, respectivamente.

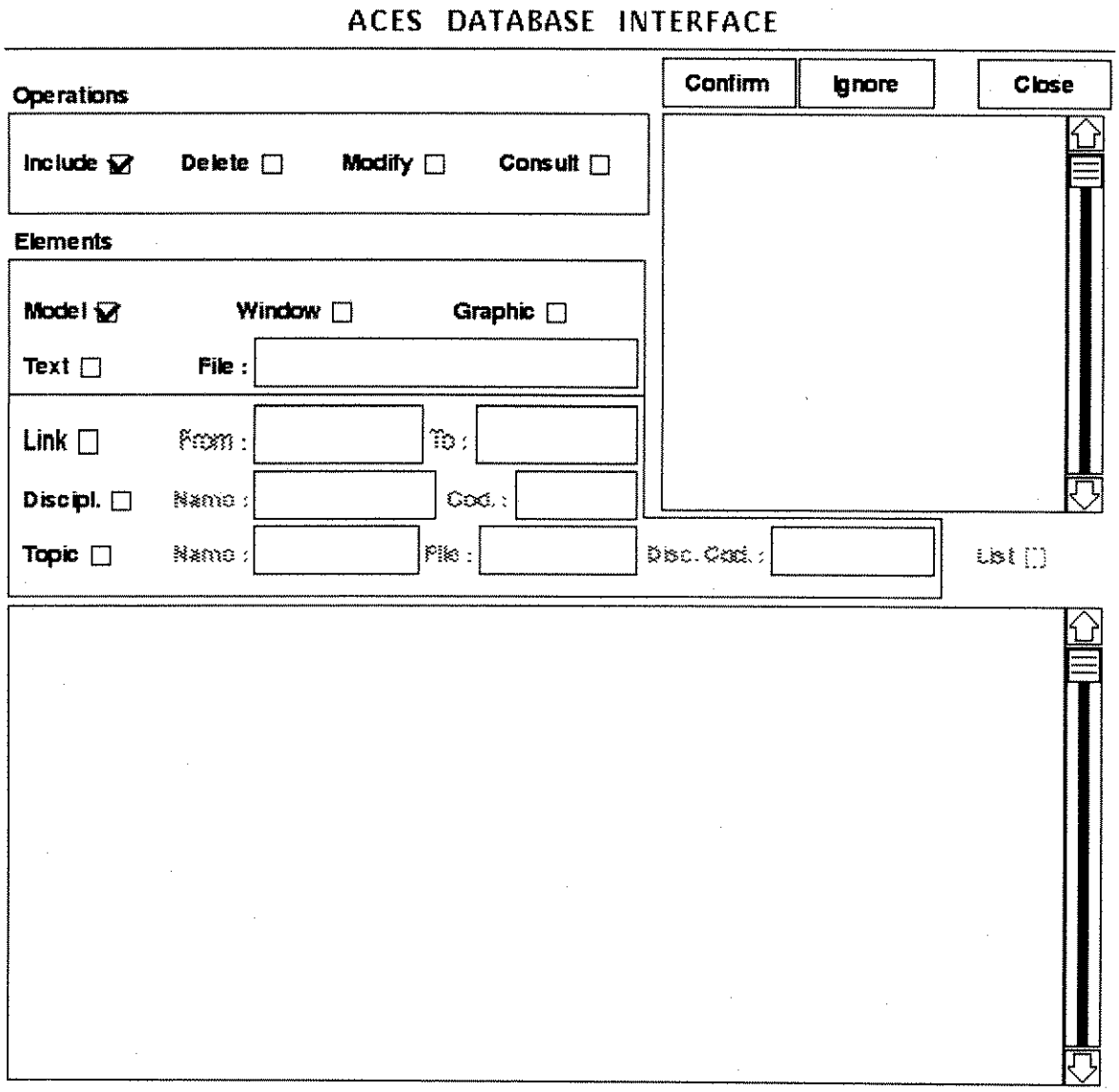

Figura 5.5: Janela apresentada pela opção **Data Base**.

As informações a serem fornecidas para um tópico são o seu nome (**Name**), o nome do arquivo (**File**) e a sigla da disciplina da qual faz parte (**Disc. Cod.**).

A opção **List**, presente também nas janelas **Users** e **Editor**, permite a exibição, na subjanela superior, de uma lista com o nome dos arquivos dos elementos da base de dados. A edição destes arquivos se processa na subjanela inferior.

Os botões **Confirm**, **Ignore** e **Close** possuem, nesta janela, as mesmas funções já descritas em seções anteriores.

5.9 Expert

O botão **Expert** faz a apresentação da janela destinada à interação com o sistema especialista.

5.10 Help

Acionando a opção **Help**, o usuário pode ter acesso aos manuais de utilização do ambiente, mostrados na forma de hipertexto pela janela representada na figura 5.6.

Esta janela contém uma grande área para exibição de texto. Acima dela estão os botões **Previous** e **Next**, encarregados respectivamente, de mostrar de forma sequencial as páginas anteriores e posteriores de um documento. A cada toque no botão **Go Back** é executada a recuperação de páginas já apresentadas. Isto é possível devido à existência de um mecanismo que armazena a seqüência de apresentação.

A janela é fechada pressionando-se o botão **Close**.

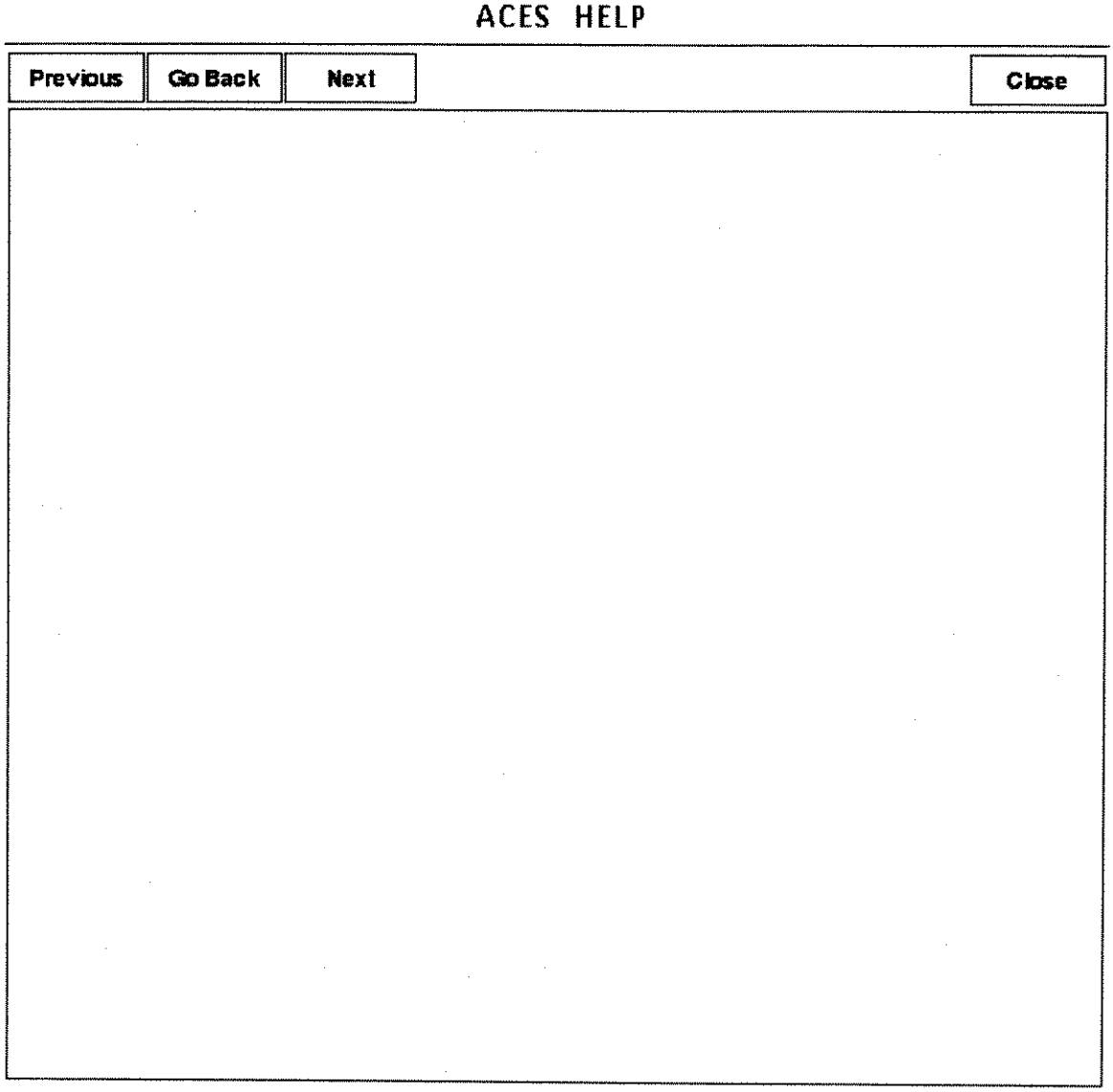

Figura 5.6: Janela do sistema de auxílio ao usuário.

5.11 Quit

A opção **Quit** é responsável pelo encerramento do programa.

5.12 Áreas de Conhecimento

Na seleção de uma área, será apresentada ao usuário uma janela para a escolha das disciplinas e tópicos desejados. Figura 5.7.

Esta janela possui uma região onde, inicialmente, aparece apenas o código da área selecionada. Neste caso, encontram-se na parte inferior, retângulos contendo as siglas das disciplinas. Através do *mouse* o usuário pode selecionar uma delas. Após esta operação, a sigla da disciplina é colocada ao lado do código da área e a relação dos tópicos é então exibida na região inferior. A partir daí, a seleção de um tópico fará a apresentação de uma nova janela (*workspace*), na qual ocorrerão as simulações.

Há ainda, na parte superior dessa janela, três botões e mais um campo para a entrada de dados.

O botão **View** fornece uma lista dos tópicos que estão relacionados entre si, ou seja, aqueles que compartilham o mesmo modelo de simulação.

Ao seu lado está o botão **Home**, que apresenta o rol dos últimos tópicos e disciplinas acessados durante o tempo em que o ambiente está sendo utilizado.

Por último, pressionando-se o botão **Goto** obtemos uma movimentação mais rápida na estrutura de diretórios, formada pelas disciplinas, com a conseqüente ativação do tópico de simulação. Para tal operação, devemos antes digitar no campo ao lado, a sigla da disciplina e o nome do tópico desejados no formato semelhante ao de um caminho de diretórios (*path*).

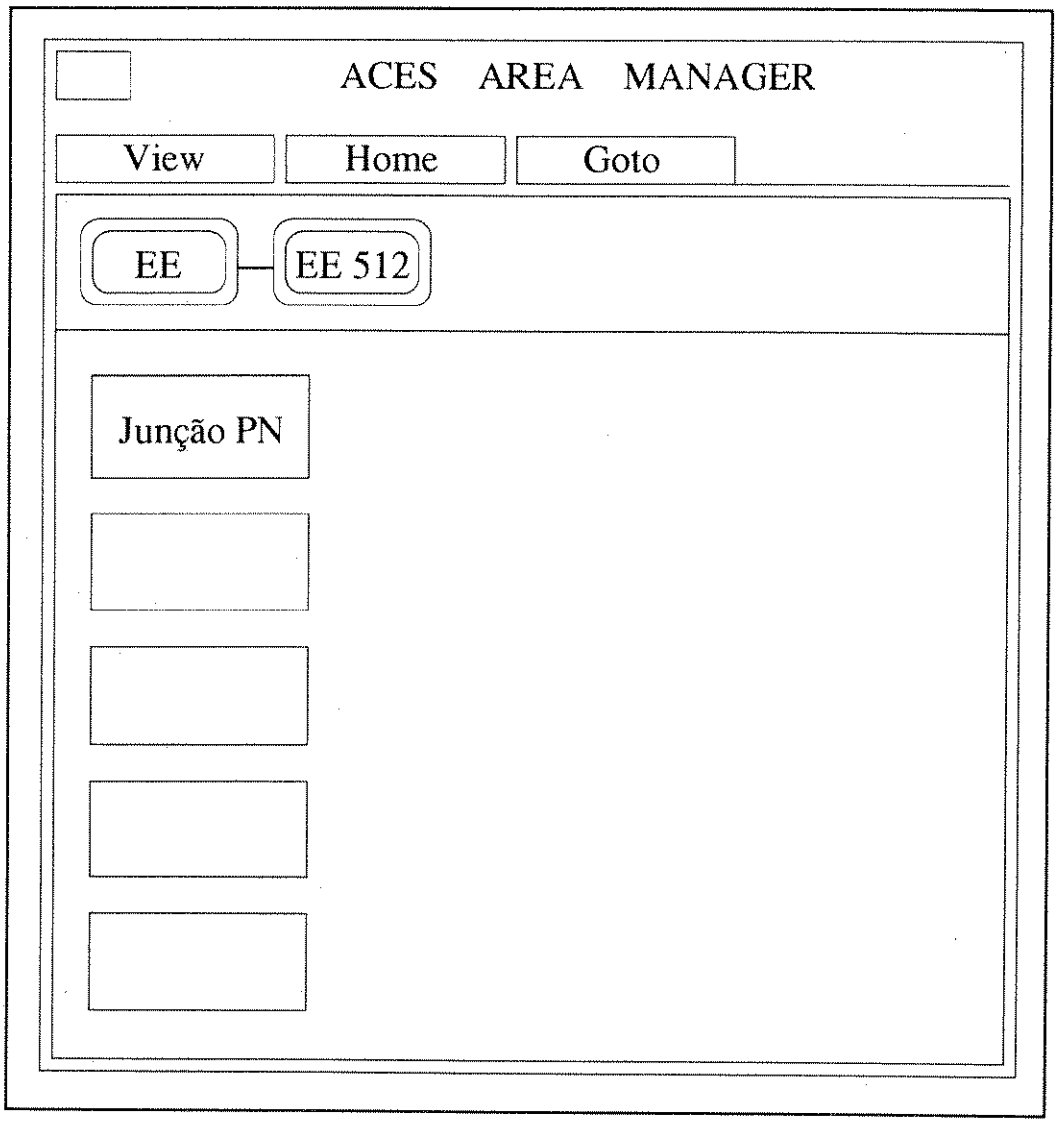

Figura 5.7: Janela para a escolha das disciplinas e tópicos de simulação.

5.13 Janela de Simulação

Uma vez definido o tópico a ser estudado, é apresentada ao usuário uma janela padrão para simulação. Figura 5.8.

Esta janela indica, no canto superior esquerdo, a disciplina e o tópico a que se refere. O botões **Graphics**, **Text** e **Windows** fazem a apresentação, respectivamente, de menus para a escolha das opções de gráficos, textos e subjanelas de interface gráfica. O botão **Print** é responsável pela ativação da impressão de gráficos e textos.

A região abaixo destes botões compreende o *workspace*, que é o lugar onde são exibidas as subjanelas e portanto, de interação com o usuário. A figura 5.9 mostra o exemplo de um *workspace* completo.

Uma observação importante refere-se ao grande número de janelas de simulação que podem estar presentes na tela da estação de trabalho. Isto pode prejudicar a correlação entre tópicos e disciplinas que estão sendo estudados pelo usuário.

Diante disso, sugerimos a abertura de uma pequena janela no momento da chamada do primeiro tópico de uma disciplina. Ela será identificada pela sigla desta e guardará o nome dos tópicos no instante em que suas janelas forem iconizadas. A figura 5.10 representa esta sugestão.

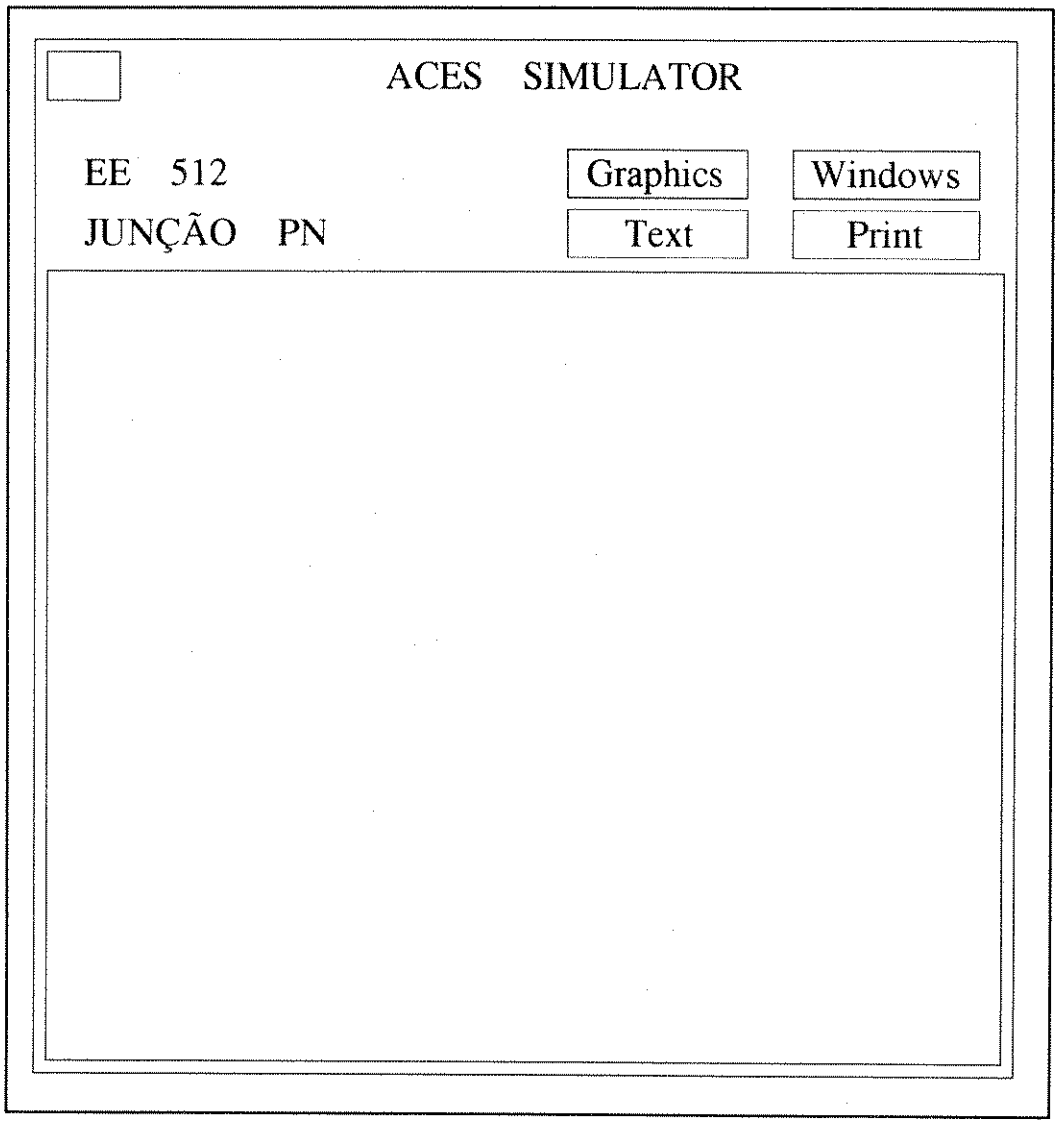

Figura 5.8: Janela para simulação.

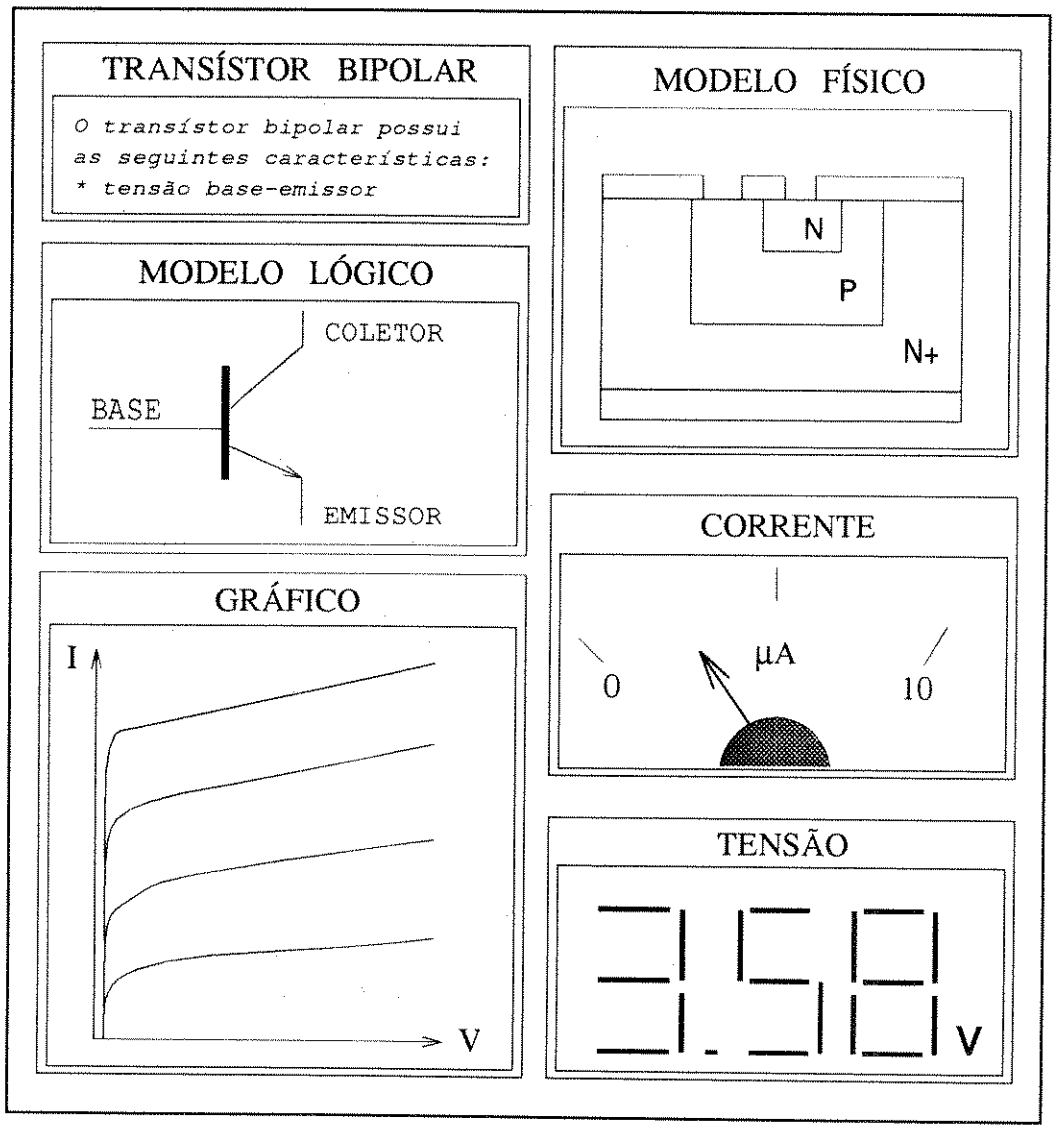

Figura 5.9: Exemplo de tela gerada durante a simulação.

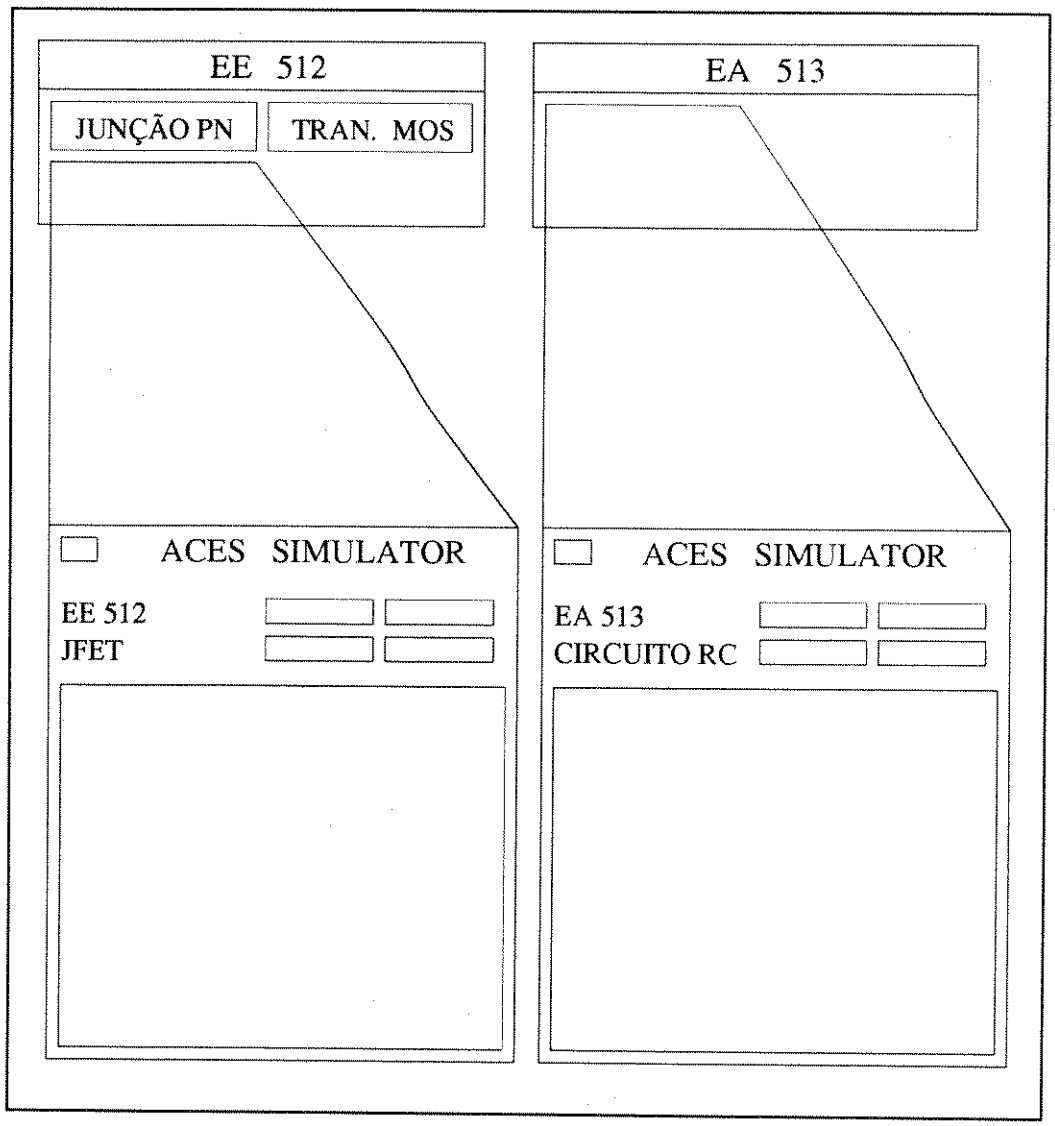

Figura 5.10: Janelas para ícones dos tópicos em cada disciplina.

Capítulo 6

Conclusão

Durante as etapas de especificação e construção do ambiente, bem como na fase de elaboração deste texto, observamos a sua grandiosidade e o vasto campo existente para sua aplicação.

Acima de tudo, este trabalho de tese vem consolidar a idéia de um projeto inovador e servir como ponto de referência para a continuação de sua implementação.

Este ambiente envolve em sua elaboração uma série de conceitos, o que o caracteriza como multidisciplinar. Não só com relação à formulação de modelos para uso, mas também durante as etapas de implementação, pessoas de diversas áreas devem ser consultadas.

A interface gráfica é o ponto que merece destaque. Na sua aplicação aos usuários, temos que estar atentos às suas reações, de maneira a comprovar a sua eficiência no campo da compreensão e da praticidade.

Reforçamos a idéia de clareza e objetividade na seqüência deste projeto. A sua flexibilidade torna-o aberto à implantação de novas características. As inovações e atualizações vindouras devem acompanhar a evolução natural da área tecnológica, estando atentas à linha-mestra aqui apresentada.

Dessa forma, podemos sugerir a introdução de recursos para a mani-

pulação de imagens dinâmicas, que venham demonstrar de maneira mais realista o comportamento dos modelos ou exibir experimentos realizados.

Além disso, a utilização de mecanismos para a gravação e reprodução de sons, que já se encontram bastante desenvolvidos nas estações de trabalho, constitui uma contribuição importante. Dessa maneira, o aluno poderia realizar as simulações acompanhando um roteiro gravado pelo professor.

Outro ponto a ressaltar é o que refere à sua portabilidade. Podemos perceber que num futuro bastante próximo, este programa possa ser utilizado em computadores pessoais, os quais venham a executar o sistema operacional UNIX e o sistema de janelas X Window. Logo, o acesso ao uso deste ambiente terá seus limites largamente ampliados.

Sem dúvida, a sua utilização irá tornar mais dinâmica a apresentação das aulas e esperamos que ele venha a cumprir os seus objetivos.

Apêndice A

Glossário

ANSI: Sigla do Instituto Americano de Padrões (*American National Standards Institute*).

Área de login: Em um sistema computacional que comporta diversos usuários, refere-se ao diretório principal de cada um deles.

Biblioteca: Conjunto de subrotinas ou funções de programas que podem ser utilizados em diversas aplicações.

Daemon: É um processo contínuo que manipula funções de um sistema operacional, tais como a administração da rede ou a fila da impressora.

Display: Tela do computador, mostrador.

Estação de trabalho: Do inglês *workstation*. É um computador, geralmente ligado em rede, com grande capacidade de processamento e que oferece importantes recursos gráficos.

Exportação: Mecanismo que possibilita a referência de sistemas de arquivos entre estações de trabalho de uma mesma rede ou de redes distintas.

GCC: *GNU C Compiler*. Compilador para a linguagem C desenvolvido por um consórcio de *software* chamado GNU.

Hardware: Nome genérico para o conjunto dos equipamentos de um sistema

computacional.

Ícone: Pequena representação em forma de imagem de uma janela.

Interface: Apresentação de um programa computacional junto ao usuário, programa que faz a comunicação entre dois outros.

Janela: Um retângulo contendo elementos de um programa.

Layout: Forma de apresentação.

Linguagem C: Linguagem de programação de computadores de finalidade geral.

Link: Referência a um arquivo ou diretório remoto.

Linkedição: Operação de reunião das funções de um programa fonte com as existentes numa biblioteca para geração do programa final executável.

Login name: Identificação de um usuário junto a um sistema operacional ou a uma rede de computadores.

Loop: Parte de um programa computacional que ao ser executada se repete um determinado número de vezes.

Mouse: Dispositivo que faz o posicionamento e a seleção de objetos mostrados na tela de um microcomputador ou de uma estação de trabalho.

On line: Expressão inglesa que significa a execução de uma determinada tarefa pelo computador ao mesmo tempo em que se está utilizando-o.

Path: Seqüência hierárquica para os nomes de diretórios.

Performance: Atuação, desempenho.

Pixmap: É um conjunto de pontos que formam uma imagem.

Portabilidade: Propriedade de um programa poder ser executado em diversos computadores ou sistemas operacionais.

Protocolo: Conjunto de regras para troca de informações entre programas em uma rede de computadores.

Residente: Programa que permanece na memória principal do computador, junto ao sistema operacional.

Servidora: Estação de trabalho de uma rede que oferece um serviço para as demais, como por exemplo, correio eletrônico.

Software: Conjunto de programas para um computador.

Status: Condição, estado, situação.

Subjanela: Janela interior a uma outra.

Toolkit: É uma biblioteca de subrotinas dedicadas a uma determinada finalidade.

UNIX: Sistema operacional utilizado em uma grande variedade de computadores, especialmente em estações de trabalho.

Widgets: Conjunto de elementos que formam uma interface com o usuário.

Workspace: Região de trabalho. Janela que comporta diversas aplicações para interação com o usuário.

Xlib: Conjunto de funções de baixo nível que fazem acesso o protocolo X e que são utilizadas para a implementação de programas que possuam uma interface gráfica.

XV: Programa para tratamento de imagens estáticas, de domínio público, desenvolvido por John Bradley, na Universidade da Pensilvânia.

X Window: Sistema de janelas que pode ser executado em diferentes modelos de computadores.

Apêndice B

Relação dos Módulos Implementados

Arquivos de cabeçalho que contêm as declarações das funções e as definições de constantes e estruturas utilizadas em cada módulo do programa.

allbuttons.h buttons.h chkbutt.h chkusr.h database.h editor.h event.h help.h icons.h mail.h main.h menu.h programs.h queryusr.h readmaster.h scroll.h setup.h users.h util.h winlist.h

Arquivo de cabeçalho que armazena os desenhos e formas utilizados pelos elementos da interface gráfica.

bitmaps.h

Faz a chamada para a criação dos botões.

allbuttons.c

Contém as funções para criação e controle dos botões.

buttons.c

Armazena as funções de criação e controle dos *check boxes*.

chkbutt.c

Módulo que faz a verificação da permissão de acesso do usuário ao ambiente.

chkusr.c

Módulo das funções da janela de acesso ao banco de dados.

database.c

Contém as funções da janela do editor.

editor.c

Controla os eventos vindos do teclado e *mouse* para o ambiente.

event.c

Possui as funções da janela do sistema de auxílio ao usuário.

help.c

Armazena as funções para criação de ícones e *pixmaps*.

icons.c

Contém as funções da janela de correio eletrônico.

mail.c

Módulo principal do programa. Faz a ativação dos demais.

main.c

Arquivo onde estão as funções para criação e controle de menus.

menu.c

Possui as funções que implementam a opção de chamada a programas externos.

programs.c

Módulo que faz acesso ao arquivo de cadastro de usuários.

queryusr.c

Módulo de leitura do arquivo das áreas e disciplinas.

readmaster.c

Contém as funções que implementam o *scrollbar*.

scroll.c

Armazena as funções relacionadas à opção **Setup**.

setup.c

Possui as funções da janela de controle de usuários.

users.c

Arquivo onde se encontram funções de utilidade geral para os outros módulos.

util.c

Módulo de criação e controle de janelas que possuem *scrollbar*.

winlist.c

Arquivo para a compilação e *linkedição* dos módulos.

Makefile

Arquivo das áreas e disciplinas do ambiente.

aces-master

Arquivo de cadastro de usuários.

aces.adm

Arquivo executável.

aces

Apêndice C

Listagem das Partes Principais do Programa

```
/*
  ================================================================================
  *
  * Projeto ACES
  *
  * Modulo principal do programa
  *
  * Arquivo : MAIN.C
  *
  * Ultima atualizacao : 11/08/93
  *
  * Ambiente SunOS 4.1.2
  *
  * Modulo compilado em GNU C (gcc)
  *
  ================================================================================
*/

 /* Arquivos de inclusao */

#include "main.h"
#include "icons.h"
```

```
#include "buttons.h"
#include "chkbutt.h"
#include "util.h"
#include "allbuttons.h"
#include "menu.h"
#include "scroll.h"
#include "winlist.h"
#include "event.h"
#include "setup.h"
#include "users.h"
#include "programs.h"
#include "mail.h"
#include "database.h"
#include "editor.h"
#include "help.h"

void main(int argc, char *argv[])

{

 /* Declaracao de variaveis */

 Display               *display;
 Window                root_win, win, icon_win;
 GC                    win_gc;
 unsigned int          border_width = 0;
 int                   screen_num;
 char                  *window_name = "ACES      MANAGER";
 char                  *icon_name = "ACES WM";
 static char           *progname;
 XSizeHints            size_hints;
 XFontStruct           *font_info, *text_font_info;
 char                  *display_name = NULL;
 Screen                *screen_ptr;
 XWindowAttributes     windowattr;
 XSetWindowAttributes  xswa;
 XWMHints              wm_hints;
 XClassHint            class_hints;
 XTextProperty         windowName, iconName;
 char                  *color_name = "#BFAA72";
```

```
char                *histr = "#F9DF9C";
char                *lostr = "#9E8C5E";
char                *icostr = "#7CD3DF";
unsigned long       bg, black, white, locol, hicol, iconcol;
XColor              panel;
Colormap            cmap;
Font                font, textfont;
Cursor              text_cursor;
char                *fname = "Helvetica-Bold12";
button              *op_butt[NOPBUTTONS], *area_butt[NAREABUTTONS];

/* Obtem o nome do programa da linha de comando */

progname = argv[0];

/* Abre o display */

if ((display = XOpenDisplay(display_name)) == NULL )
  {
    (void)fprintf(stderr,"%s : cannot connect to X server %s \n",progname,
                  XDisplayName(display_name));
    exit(-1);
  }
screen_num = DefaultScreen(display);
screen_ptr = DefaultScreenOfDisplay(display);

root_win = RootWindow(display, screen_num);

/* Seleciona o colormap default */

cmap = DefaultColormap(display, screen_num);

if (XGetWindowAttributes(display, root_win, &windowattr) == 0)
  {
    fprintf(stderr, "%s; failed to get window attributes. \n", progname);
    exit(-1);
  }

 /* Alocacao das cores branca e preta */
```

```
 white = WhitePixel(display,screen_num);
 black = BlackPixel(display,screen_num);

/* Inicializa o modulo util.c */

Init_Util(display, root_win, black, white);

/* Alocacao das outras cores */

panel = Alloc_Colors(cmap, color_name, progname);
bg = panel.pixel;

panel = Alloc_Colors(cmap, histr, progname);
hicol = panel.pixel;

panel = Alloc_Colors(cmap, lostr, progname);
locol = panel.pixel;

panel = Alloc_Colors(cmap, icostr, progname);
iconcol = panel.pixel;

/* Cria a janela principal e a janela de icone */

win = XCreateSimpleWindow(display, root_win, 0, 0, MAIN_WIDTH, MAIN_HEIGHT,
  border_width, black, bg);

icon_win = XCreateSimpleWindow(display, root_win, 0, 0, ICON_WIDTH, \
ICON_HEIGHT, border_width, black, bg);

/* Cria o cursor para edicao de textos */

text_cursor = XCreateFontCursor(display, XC_xterm);

/* Cria o Graphics Context (GC) */

win_gc = XCreateGC(display, win, 0, 0);
XSetBackground(display, win_gc, bg);
XSetForeground(display, win_gc, black);

/* Seleciona os tipos de letras a serem usados na janela principal */
```

```
font_info = Set_Fonts(fname, progname);
font = font_info->fid;
XSetFont(display, win_gc, font);

/* Inicializa os outros modulos */

Init_All_Buttons(win, black, bg);
InitChkButtons(display, root_win, win_gc, black, bg);
Init_Menu(display, root_win, bg, black);
InitScroll(display, win_gc, root_win, black, bg);
InitWinList(display, root_win, black, white, bg);

/* Cria o icone colorido */

Main_Icon(display, icon_win, progname, hicol, locol, iconcol);

/* Apresenta o nome da janela principal */

if (XStringListToTextProperty(&window_name, 1, &windowName) == 0)
  {
    (void)fprintf(stderr, "%s: structure allocation for windowName failed.\n", \
  progname);
    exit(-1);
  }

/* Apresenta o nome da janela de icone */

if (XStringListToTextProperty(&icon_name, 1, &iconName) == 0)
  {
    (void)fprintf(stderr, "%s: structure allocation for iconName failed. \n", \
  progname);
    exit(-1);
  }

/* Ajusta os parametros de tamanho e posicao da janela principal */

size_hints.flags = PPosition | PSize | PMinSize | PMaxSize;
size_hints.width = size_hints.min_width = size_hints.max_width = MAIN_WIDTH;
size_hints.height = size_hints.min_height = size_hints.max_height = MAIN_HEIGHT;
```

```
size_hints.x = 10;
size_hints.y = 10;

/* Ajusta outros parametros importantes da janela principal */

wm_hints.initial_state = NormalState;
wm_hints.input = True;
wm_hints.icon_window = icon_win;
wm_hints.icon_x = 90;
wm_hints.icon_y = 5;
wm_hints.flags |= StateHint | IconPixmapHint | InputHint | IconWindowHint | \
                  IconPositionHint;

XStringListToTextProperty(&window_name, 1, &windowName);
XStringListToTextProperty(&icon_name, 1, &iconName);

xswa.backing_store = WhenMapped;
XChangeWindowAttributes(display, win, CWBackingStore, &xswa);

class_hints.res_name = progname;
class_hints.res_class = "Basicwin";

XSetStandardProperties(display, win, window_name, icon_name, icon_win, argv, \
argc, &size_hints);
XSetWMProperties(display, win, &windowName, &iconName, argv, argc, \
 &size_hints, &wm_hints, &class_hints);

StoreDeleteWindowProp(win);

/* Inicializa os botoes da janela principal */

InitButtons(display, screen_num, win_gc, root_win, cmap, black, \
    bg, hicol, locol, black, white, font_info);

/* Seleciona os eventos a serem considerados */

XSelectInput(display, win, ButtonPressMask | ExposureMask | \
     StructureNotifyMask | OwnerGrabButtonMask | EnterWindowMask | \
     LeaveWindowMask);
```

```
XSelectInput(display, root_win, SubstructureNotifyMask);

/* Desenha os botoes de opcoes */

Main_Op_Buttons(op_butt, progname);

/* Cria os pixmaps dos botoes das areas */

Make_Pixmaps(display, win, black, bg, progname);

/* Cria os botoes das areas */

Areas_Buttons(area_butt, progname);

/*
  Faz a verificacao da permissao de acesso do usuario
  e atualiza a condicao dos botoes de opcoes e das areas
*/

CheckUser(op_butt, area_butt);

/* Inicializa o modulo de eventos */

Init_Event(display, win, root_win, win_gc, text_cursor, op_butt, area_butt, \
  hicol, locol);

/* Inicializa as janelas subsidiarias */

Init_Setup(display, root_win, win, bg, black, locol);
Init_Users(display, root_win, locol, bg, black, white);
Init_Programs(display, win, bg, black);
Init_Mail(display, root_win, locol, bg, black, white);
Init_Ed(display, root_win, locol, bg, black, white);
Init_Db(display, root_win, locol, bg, black, white);
Init_Help(display, root_win, locol, bg, black, white);

/* Mapeia a janela principal e as suas subjanelas */

XMapSubwindows(display, win);
```

```
  XMapWindow(display, win);

  Setup_Menu();
  Programs_Menu();

  /* Ativa a funcao controladora de eventos */

  handle_events();

  /* Libera o GC, destroi a janela principal e fecha o display */

  XFreeGC(display, win_gc);
  XDestroyWindow(display, win);
  XCloseDisplay(display);
}

 /*
   ============================================================================
   *
   * Projeto ACES
   *
   * Modulo de controle de eventos
   *
   * Arquivo : EVENT.C
   *
   * Ultima atualizacao : 11/08/93
   *
   * Ambiente SunOS 4.1.2
   *
   * Modulo compilado em GNU C (gcc)
   *
   ============================================================================
 */

 /* Arquivos de inclusao */

#include "main.h"
#include "buttons.h"
#include "chkbutt.h"
#include "util.h"
```

```
#include "menu.h"
#include "scroll.h"
#include "winlist.h"
#include "setup.h"
#include "users.h"
#include "programs.h"
#include "mail.h"
#include "database.h"
#include "editor.h"
#include "help.h"
#include "event.h"

 /* Declaracao das variaveis locais estaticas */

static Display          *display;
static Window           root_win, win, users_win, mail_win, ed_win, db_win,
                        help_win, acsed_win, acsible_win, mailfilewin, mailtowin,
                        mailtowin, mailsubjwin, usersnamewin, usersareawin,
                        usersdiscwin, usersstatwin, edtopnamewin, edtopfilewin,
                        edmodnamewin, edmodfilewin, edwinnamewin, edwinfilewin,
                        edwinvarwin, edgraphnamewin, edgraphfilewin, edgraphvarwin,
                        edtxtnamewin, edtxtfilewin, edtxtkeywin, dbtxtfilewin,
                        dbfromlinkwin, dbtolinkwin, dbdinamewin, dbdicodwin,
                        dbtpnamewin, dbtpfilewin, dbtpdicodwin;
static GC               win_gc, users_gc, mail_gc, ed_gc, db_gc, help_gc;
static Cursor           text_cursor;
static button           **opbtlist, **areabtlist, **usersbtlist, **mailbtlist,
                        **edbtlist, **dbbtlist, **helpbtlist, **acsedbtlist,
                        **acsiblebtlist;
static unsigned int     usersUp = False, mailUp = False, edUp = False,
                        dbUp = False, helpUp = False, acsedUp = False,
                        acsibleUp = False;
static unsigned int     wasusersUp = False, wasmailUp = False, wasedUp = False,
                        wasdbUp = False, washelpUp = False, wasacsedUp = False,
                        wasacsibleUp = False;
static unsigned long    hicol, locol;
static chkbutt          *usoper_butt, *uscat_butt, *uslist_butt, *edlist_butt,
                        *dboper_butt, *dbelem_butt, *dblist_butt;
static menubutt         *setup_menu, *usbytime_menu;
static scroll           *usersshwscrl, *mailcmpscrl, *edtopicscrl, *dbworkscrl;
```

```
static winlist          *usershintwinl, *usersshwin, *mailmsgwinl, *edauxwinl,
                        *edwordswinl, *dbpreswinl;
static int              dbnumelem;

 /* Inicializa as variaveis estaticas do modulo event.c */

void Init_Event(Display *disp, Window mainw, Window rootw, GC maingc, \
Cursor txtcursor, button* oplist[], button* arealist[], \
unsigned long high_color, unsigned long low_color)

{
   display = disp;
   win = mainw;
   root_win = rootw;
   win_gc = maingc;
   text_cursor = txtcursor;
   opbtlist = oplist;
   areabtlist = arealist;
   hicol = high_color;
   locol = low_color;
}

 /* Importa os valores das variaveis do modulo setup.c */

void Setup_to_Event(Window acsedw, Window acsiblew, menubutt *setupmenu, \
    button *acsedlist[], button *acsiblelist[])

{
  acsed_win = acsedw;
  acsible_win = acsiblew;
  acsedbtlist = acsedlist;
  acsiblebtlist = acsiblelist;
  setup_menu = setupmenu;
}

 /* Importa os valores das variaveis do modulo users.c */

void Users_to_Event(Window usersw, GC usersgc, button* userslist[], \
    chkbutt *operbutt, chkbutt *catbutt, chkbutt *listbutt, \
    menubutt *bytime_menu, winlist *showwin, \
```

```
    winlist *hintswinl, Window namewin, Window areawin, \
    Window discwin, Window statwin)
{
   users_win = usersw;
   users_gc = usersgc;
   usersbtlist = userslist;
   usoper_butt = operbutt;
   uscat_butt = catbutt;
   uslist_butt = listbutt;
   usbytime_menu = bytime_menu;
   usersshwin = showwin;
   usershintwinl = hintswinl;
   usersnamewin = namewin;
   usersareawin = areawin;
   usersdiscwin = discwin;
   usersstatwin = statwin;
   XDefineCursor(display, usersnamewin, text_cursor);
   XDefineCursor(display, usersareawin, text_cursor);
   XDefineCursor(display, usersdiscwin, text_cursor);
   XDefineCursor(display, usersstatwin, text_cursor);
}

 /* Importa os valores das variaveis do modulo mail.c */

void Mail_to_Event(Window mailw, GC mailgc, button* maillist[], \
   scroll *compscroll, winlist *mesgwinl, Window filewin, \
   Window towin, Window subjwin)

{
   mail_win = mailw;
   mail_gc = mailgc;
   mailbtlist = maillist;
   mailcmpscrl = compscroll;
   mailmsgwinl = mesgwinl;
   mailfilewin = filewin;
   mailtowin = towin;
   mailsubjwin = subjwin;
   XDefineCursor(display, mailfilewin, text_cursor);
   XDefineCursor(display, mailtowin, text_cursor);
   XDefineCursor(display, mailsubjwin, text_cursor);
```

```
}

 /* Importa os valores das variaveis do modulo editor.c */

void Ed_to_Event(Window edw, GC edgc, button* edlist[], chkbutt *listbutt, \
 scroll *topicscroll, winlist *auxwinl, winlist *wordswinl, \
 Window topnamewin, Window topfilewin, Window modnamewin, \
 Window modfilewin, Window winnamewin, Window winfilewin, \
 Window winvarwin, Window graphnamewin, Window graphfilewin, \
 Window graphvarwin, Window txtnamewin, Window txtfilewin, \
 Window txtkeywin)
{
   ed_win = edw;
   ed_gc = edgc;
   edbtlist = edlist;
   edlist_butt = listbutt;
   edtopicscrl = topicscroll;
   edauxwinl = auxwinl;
   edwordswinl = wordswinl;
   edtopnamewin = topnamewin;
   edtopfilewin = topfilewin;
   edmodnamewin = modnamewin;
   edmodfilewin = modfilewin;
   edwinnamewin = winnamewin;
   edwinfilewin = winfilewin;
   edwinvarwin = winvarwin;
   edgraphnamewin = graphnamewin;
   edgraphfilewin = graphfilewin;
   edgraphvarwin = graphvarwin;
   edtxtnamewin = txtnamewin;
   edtxtfilewin = txtfilewin;
   edtxtkeywin = txtkeywin;
   XDefineCursor(display, edtopnamewin, text_cursor);
   XDefineCursor(display, edtopfilewin, text_cursor);
   XDefineCursor(display, edmodnamewin, text_cursor);
   XDefineCursor(display, edmodfilewin, text_cursor);
   XDefineCursor(display, edwinnamewin, text_cursor);
   XDefineCursor(display, edwinfilewin, text_cursor);
   XDefineCursor(display, edwinvarwin, text_cursor);
   XDefineCursor(display, edgraphnamewin, text_cursor);
```

```
   XDefineCursor(display, edgraphfilewin, text_cursor);
   XDefineCursor(display, edgraphvarwin, text_cursor);
   XDefineCursor(display, edtxtnamewin, text_cursor);
   XDefineCursor(display, edtxtfilewin, text_cursor);
   XDefineCursor(display, edtxtkeywin, text_cursor);
}

 /* Importa os valores das variaveis do modulo database.c */

void Db_to_Event(Window dbw, GC dbgc, button* dblist[], chkbutt *operbutt, \
 chkbutt *elembutt, chkbutt *listbutt, scroll *workscroll, \
 winlist *preswinl, Window txtfilewin, Window fromlinkwin, \
                 Window tolinkwin, Window dinamewin, Window dicodwin, \
 Window tpnamewin, Window tpfilewin, Window tpdicodwin)

{
   db_win = dbw;
   db_gc = dbgc;
   dbbtlist = dblist;
   dboper_butt = operbutt;
   dbelem_butt = elembutt;
   dblist_butt = listbutt;
   dbworkscrl = workscroll;
   dbpreswinl = preswinl;
   dbtxtfilewin = txtfilewin;
   dbfromlinkwin = fromlinkwin;
   dbtolinkwin = tolinkwin;
   dbdinamewin = dinamewin;
   dbdicodwin = dicodwin;
   dbtpnamewin = tpnamewin;
   dbtpfilewin = tpfilewin;
   dbtpdicodwin = tpdicodwin;
   XDefineCursor(display, dbtxtfilewin, text_cursor);
   XDefineCursor(display, dbfromlinkwin, text_cursor);
   XDefineCursor(display, dbtolinkwin, text_cursor);
   XDefineCursor(display, dbdinamewin, text_cursor);
   XDefineCursor(display, dbdicodwin, text_cursor);
   XDefineCursor(display, dbtpnamewin, text_cursor);
   XDefineCursor(display, dbtpfilewin, text_cursor);
   XDefineCursor(display, dbtpdicodwin, text_cursor);
```

```
}

 /* Importa os valores das variaveis do modulo help.c */

void Help_to_Event(Window helpw, GC helpgc, button* helplist[])

{
   help_win = helpw;
   help_gc = helpgc;
   helpbtlist = helplist;
}

 /* Faz o controle dos botoes da janela principal */

void Pointer_In_Main(XEvent *event, int *done, int x, int y)

{
  int op, area;

  op = ButtonClicked(x,y,opbtlist,10);
  op--;
  area = ButtonClicked(x,y,areabtlist,5);
  area--;

  /* Faz a escolha entre os botoes de opcoes */

  switch (op)
    {
      case SETUP    : Handle_Setup();
                      break;
      case USERS    : XMapSubwindows(display, users_win);
                      XUnmapWindow(display, usbytime_menu->menuwin);
                      XMapRaised(display, users_win);
                      usersUp = True;
                      break;
      case PROGRAMS : Handle_Programs();
                      break;
      case MAIL     : XMapSubwindows(display, mail_win);
                      XMapRaised(display, mail_win);
                      mailUp = True;
```

```
                    break;
      case EXAMPLE  : break;
      case EDITOR   : XMapSubwindows(display, ed_win);
                    XMapRaised(display, ed_win);
                    edUp = True;
                    break;
      case DATABASE : XMapSubwindows(display, db_win);
                    XMapRaised(display, db_win);
                    dbUp = True;
                    break;
      case EXPERT   : break;
      case HELP     : XMapSubwindows(display, help_win);
                    XMapRaised(display, help_win);
                    helpUp = True;
                    break;
      case QUIT     : *done = 1;
                    break;
   }

 /* Faz a escolha entre os botoes das areas */

 switch (area)
   {
     case ELETRICA: break;
     case MECANICA: break;
     case ENGQUIMICA: break;
     case FISICA: break;
     case QUIMICA: break;
   }
}

 /* Faz o controle do menu da opcao Setup */

void Handle_Setup()

{
   int    sel_option;

   sel_option = TrackMenu(setup_menu);
   switch (sel_option)
```

```
    {
      case 0: break;
      case 1: XMapSubwindows(display, acsed_win);
              XMapRaised(display, acsed_win);
              acsedUp = True;
              break;
      case 2: XMapSubwindows(display, acsible_win);
              XMapRaised(display, acsible_win);
              acsibleUp = True;
              break;
    }
}

 /* Controla o ponteiro na janela Accessed Disciplines */

void Pointer_In_Acsed(XEvent *event, Time lasttime, int x, int y)

{
   int  acsed;

   acsed = ButtonClicked(x, y, acsedbtlist, 1);
   acsed--;
   if (acsed == 0)
     {
       XUnmapWindow(display, acsed_win);
       acsedUp = False;
     }
   else
     if PTINRECT(x, y, 0, 0, 300, 20)
       if (event->xbutton.time - lasttime >= DBLCLKTIME)
         move_window();
}

 /* Controla o ponteiro na janela Accessible Disciplines */

void Pointer_In_Acsible(XEvent *event, Time lasttime, int x, int y)

{
   int  acsible;
```

```
    acsible = ButtonClicked(x, y, acsiblebtlist, 1);
    acsible--;
    if (acsible == 0)
      {
        XUnmapWindow(display, acsible_win);
        acsibleUp = False;
      }
    else
      if PTINRECT(x, y, 0, 0, 300, 20)
        if (event->xbutton.time - lasttime >= DBLCLKTIME)
          move_window();
}

 /* Controla o ponteiro na janela Users */

void Pointer_In_Users(XEvent *event, Time lasttime, int x, int y)

{
  int users, numoper, numcat;

  if ((numoper = ClickChkList(usoper_butt, x, y)) >= 0)
    {
      TrackChkList(usoper_butt, numoper);
      if (numoper == 0)
        UpdateChkButtonStatus(uslist_butt, INACTIVE);
      else
        UpdateChkButtonStatus(uslist_butt, ACTIVE);
    }
  else
    if ((numcat = ClickChkList(uscat_butt, x, y)) >= 0)
     TrackChkList(uscat_butt, numcat);
    else
      if (ClickChkButton(uslist_butt, x, y))
        TrackChkButton(uslist_butt);
      else
        if PTINRECT(x, y, 574, 370, 20, 20)
          TrackScroll(usersshwin->scrl, x, y, 0);
        else
          if PTINRECT(x, y, 574, 562, 20, 20)
    TrackScroll(usersshwin->scrl, x, y, 2);
```

```
          else
            if PTINRECT(x, y, 574, 65, 20, 20)
              TrackScroll(usershintwinl->scrl, x, y, 0);
            else
              if PTINRECT(x, y, 574, 257, 20, 20)
                TrackScroll(usershintwinl->scrl, x, y, 2);
              else
                {
                  users = ButtonClicked(x, y, usersbtlist, 10);
                  users--;
                  switch (users)
                    {
                      case USERSCONFIRM : break;
                      case USERSIGNORE  : break;
                      case BYTIME       : Handle_ByTime();
                                          break;
                      case BYPATH       : break;
                      case ALLUSERS     : break;
                      case USERSBYDISC  : break;
                      case DISCIPLINES  : Show_All_Disciplines();
                  break;
                      case LOGUSERS     : break;
      case USERSCLEAR   : Clear_Show_Window();
                  break;
                      case USERSCLOSE   : XUnmapWindow(display, users_win);
                                          usersUp = False;
                                          break;
                      default           : if PTINRECT(x, y, 0, 0, 600, 30)
                                            if (event->xbutton.time - \
lasttime >= DBLCLKTIME)
                                              move_window();
                    }
                }
}

 /* Controla o ponteiro na janela Mail */

void Pointer_In_Mail(XEvent *event, Time lasttime, int x, int y)

{
```

```
  int mail;

  if PTINRECT(x, y, 574, 65, 20, 20)
    TrackScroll(mailmsgwinl->scrl, x, y, 0);
  else
    if PTINRECT(x, y, 574, 227, 20, 20)
      TrackScroll(mailmsgwinl->scrl, x, y, 2);
    else
      if PTINRECT(x, y, 574, 330, 20, 20)
        TrackScroll(mailcmpscrl, x, y, 0);
      else
        if PTINRECT(x, y, 574, 562, 20, 20)
          TrackScroll(mailcmpscrl, x, y, 2);
        else
          {
            mail = ButtonClicked(x, y, mailbtlist, 6);
            mail--;
            switch (mail)
              {
                case MAILDELETE : break;
                case SAVE       : break;
                case REPLY      : break;
                case DELIVER    : break;
                case MAILCLEAR  : break;
                case MAILCLOSE  : XUnmapWindow(display, mail_win);
                                  mailUp = False;
                                  break;
                default         : if PTINRECT(x, y, 0, 0, 600, 30)
                                    if (event->xbutton.time -  \
lasttime >= DBLCLKTIME)
                                      move_window();
              }
          }
}

 /* Controla o ponteiro na janela Editor */

void Pointer_In_Ed(XEvent *event, Time lasttime, int x, int y)

{
```

```
int ed;

if PTINRECT(x, y, 574, 65, 20, 20)
  TrackScroll(edauxwin1->scrl, x, y, 0);
else
  if PTINRECT(x, y, 574, 197, 20, 20)
    TrackScroll(edauxwin1->scrl, x, y, 2);
  else
    if PTINRECT(x, y, 574, 240, 20, 20)
      TrackScroll(edwordswin1->scrl, x, y, 0);
    else
      if PTINRECT(x, y, 574, 337, 20, 20)
        TrackScroll(edwordswin1->scrl, x, y, 2);
      else
        if PTINRECT(x, y, 574, 365, 20, 20)
          TrackScroll(edtopicscrl, x, y, 0);
        else
          if PTINRECT(x, y, 574, 562, 20, 20)
            TrackScroll(edtopicscrl, x, y, 2);
          else
            if (ClickChkButton(edlist_butt, x, y))
              TrackChkButton(edlist_butt);
            else
              {
                ed = ButtonClicked(x, y, edbtlist, 4);
                ed--;
                switch (ed)
                  {
                    case EDCONFIRM : break;
                    case EDIGNORE  : break;
                    case EDDELETE  : break;
                    case EDCLOSE   : XUnmapWindow(display, ed_win);
                             edUp = False;
                                    break;
                    default        : if PTINRECT(x, y, 0, 0, 600, 30)
                                       if (event->xbutton.time - \
  lasttime >= DBLCLKTIME)
                                         move_window();
                  }
              }
```

```
}

 /* Controla o ponteiro na janela Database */

void Pointer_In_Db(XEvent *event, Time lasttime, int x, int y)

{
  int db, numoper;

  if ((numoper = ClickChkList(dboper_butt, x, y)) >= 0)
    {
      TrackChkList(dboper_butt, numoper);
      if (numoper == 0)
        UpdateChkButtonStatus(dblist_butt, INACTIVE);
      else
        UpdateChkButtonStatus(dblist_butt, ACTIVE);
    }
  else
    if ((dbnumelem = ClickChkList(dbelem_butt, x, y)) >= 0)
      {
        TrackChkList(dbelem_butt, dbnumelem);
        Db_Strings();
      }
    else
      if (ClickChkButton(dblist_butt, x, y))
        TrackChkButton(dblist_butt);
      else
        if PTINRECT(x, y, 574, 325, 20, 20)
          TrackScroll(dbworkscrl, x, y, 0);
        else
          if PTINRECT(x, y, 574, 562, 20, 20)
            TrackScroll(dbworkscrl, x, y, 2);
          else
            if PTINRECT(x, y, 574, 65, 20, 20)
              TrackScroll(dbpreswinl->scrl, x, y, 0);
            else
              if PTINRECT(x, y, 574, 254, 20, 20)
                TrackScroll(dbpreswinl->scrl, x, y, 2);
              else
                {
```

```
                db = ButtonClicked(x, y, dbbtlist, 3);
                db--;
                switch (db)
                  {
                    case DBCONFIRM : break;
                    case DBIGNORE  : break;
                    case DBCLOSE   : XUnmapWindow(display, db_win);
                                     dbUp = False;
                                     break;
                    default        : if PTINRECT(x, y, 0, 0, 600, 30)
                                       if (event->xbutton.time - \
    lasttime >= DBLCLKTIME)
                                         move_window();
                  }
              }
}

 /* Controla o ponteiro na janela Help */

void Pointer_In_Help(XEvent *event, Time lasttime, int x, int y)

{
  int help;

  help = ButtonClicked(x, y, helpbtlist, 4);
  help--;
  switch (help)
    {
      case PREVIOUS  : break;
      case GOBACK    : break;
      case NEXT      : break;
      case HELPCLOSE : XUnmapWindow(display, help_win);
                       helpUp = False;
                       break;
      default        : if PTINRECT(x, y, 0, 0, 600, 30)
                         if (event->xbutton.time - lasttime >= DBLCLKTIME)
                           move_window();
    }
}
```

```
/*
   Loop de eventos. Faz a verificacao de
   todos os eventos que chegam ao programa
*/

void handle_events()

{
   XEvent event;
   int done = 0;
   static Time lasttime = 0;

    while (done == 0)
      {

       /* Obtem o proximo evento */

       XNextEvent(display,&event);
       switch (event.type)
         {

            /*
       Entra nesta opcao quando e necessaria a
               exibicao de uma janela ou de seus componentes
            */

            case Expose:
               if (event.xexpose.count == 0)
                 {
           if (event.xbutton.window == win)
    {
                       RedrawButtons(opbtlist,10);
                       RedrawButtons(areabtlist,5);
                       XDrawLine(display, win, win_gc, 0, 0, MAIN_WIDTH, 0);
                       XDrawLine(display, win, win_gc, 0, 65, MAIN_WIDTH, 65);
           }
                 if (event.xbutton.window == users_win)
                   Users_Draw();
   if (event.xbutton.window == mail_win)
                   Mail_Draw();
```

```
                if (event.xbutton.window == ed_win)
                  Ed_Draw();
        if (event.xbutton.window == db_win)
                  Db_Draw();
    if (event.xbutton.window == help_win)
                  Help_Draw();
      if (event.xbutton.window == acsed_win)
                  Acsed_Draw();
     if (event.xbutton.window == acsible_win)
                  Acsible_Draw();
              }
           break;

  /* Entra nesta opcao quando o botao do mouse e pressionado */

         case ButtonPress:
            {
               int  x, y, i;

               x = event.xbutton.x;
               y = event.xbutton.y;

switch (event.xbutton.button)
  {

    /* Botao esquerdo do mouse */

    case Button1:
                      if (event.xbutton.window == win)
                        Pointer_In_Main(&event, &done, x, y);
                      if (event.xbutton.window == users_win)
                        Pointer_In_Users(&event, lasttime, x, y);
        if (event.xbutton.window == mail_win)
                        Pointer_In_Mail(&event, lasttime, x, y);
                      if (event.xbutton.window == ed_win)
                        Pointer_In_Ed(&event, lasttime, x, y);
                      if (event.xbutton.window == db_win)
                        Pointer_In_Db(&event, lasttime, x ,y);
                      if (event.xbutton.window == help_win)
  Pointer_In_Help(&event, lasttime, x, y);
```

```
if (event.xbutton.window == acsed_win)
  Pointer_In_Acsed(&event, lasttime, x, y);
if (event.xbutton.window == acsible_win)
  Pointer_In_Acsible(&event, lasttime, x, y);
if (event.xbutton.window == usersshwin->scrl->win)
                          TrackScroll(usersshwin->scrl, x, y, -1);
if (event.xbutton.window == mailcmpscrl->win)
                          TrackScroll(mailcmpscrl, x, y, -1);
if (event.xbutton.window == edtopicscrl->win)
                          TrackScroll(edtopicscrl, x, y, -1);
if (event.xbutton.window == dbworkscrl->win)
                          TrackScroll(dbworkscrl, x, y, -1);
if (event.xbutton.window == usershintwinl->scrl->win)
  TrackScroll(usershintwinl->scrl, x, y, -1);
if (event.xbutton.window == mailmsgwinl->scrl->win)
  TrackScroll(mailmsgwinl->scrl, x, y, -1);
if (event.xbutton.window == edauxwinl->scrl->win)
  TrackScroll(edauxwinl->scrl, x, y, -1);
if (event.xbutton.window == edwordswinl->scrl->win)
  TrackScroll(edwordswinl->scrl, x, y, -1);
if (event.xbutton.window == dbpreswinl->scrl->win)
  TrackScroll(dbpreswinl->scrl, x, y, -1);
if (event.xbutton.window == usershintwinl->win)
                          {
    i = WinListClick(usershintwinl, &event.xbutton);
    if ( i != -1 )
                              SelectOption(usershintwinl, i);
                          }
if (event.xbutton.window == mailmsgwinl->win)
  {
    i = WinListClick(mailmsgwinl, &event.xbutton);
                            if ( i != -1 )
      SelectOption(mailmsgwinl, i);
  }
if (event.xbutton.window == edauxwinl->win)
  {
    i = WinListClick(edauxwinl, &event.xbutton);
                            if ( i != -1 )
      SelectOption(edauxwinl, i);
  }
```

```
if (event.xbutton.window == edwordswinl->win)
  {
    i = WinListClick(edwordswinl, &event.xbutton);
                              if ( i != -1 )
      SelectOption(edwordswinl, i);
  }
if (event.xbutton.window == dbpreswinl->win)
  {
    i = WinListClick(dbpreswinl, &event.xbutton);
                              if ( i != -1 )
      SelectOption(dbpreswinl, i);
  }
          }
               break;
            }

  /* Opcao executada no encerramento do programa */

         case ClientMessage:
            {
               Atom proto, delwin;

               proto = XInternAtom(display, "WM_PROTOCOLS", FALSE);
               delwin = XInternAtom(display, "WM_DELETE_WINDOW", FALSE);

               if (event.xclient.message_type == proto &&  \
    event.xclient.data.l[0] == delwin)
                  if (event.xclient.window == win) exit(0);
               break;
            }

  /* Entra nesta opcao quando uma janela esta mapeada */

  case MapNotify:
              if (event.xmap.window == win)
                {
   if (wasusersUp)
     TheUsersWin(wasusersUp, &usersUp, &wasusersUp);
                  if (wasmailUp)
     TheMailWin(wasmailUp, &mailUp, &wasmailUp);
```

```
  if (wasedUp)
    TheEdWin(wasedUp, &edUp, &wasedUp);
  if (wasdbUp)
     TheDbWin(wasdbUp, &dbUp, &wasdbUp);
  if (washelpUp)
    TheHelpWin(washelpUp, &helpUp, &washelpUp);
  if (wasacsedUp)
                    TheAcsedWin(wasacsedUp, &acsedUp, &wasacsedUp);
  if (wasacsibleUp)
                    TheAcsibleWin(wasacsibleUp, &acsibleUp, &wasacsibleUp);
}
     break;

  /* Entra nesta opcao quando uma janela nao esta mapeada */


          case UnmapNotify:
              if (event.xunmap.window == win)
                {
  if (usersUp)
    TheUsersWin(False, &usersUp, &wasusersUp);
                  if (mailUp)
    TheMailWin(False, &mailUp, &wasmailUp);
  if (edUp)
    TheEdWin(False, &edUp, &wasedUp);
  if (dbUp)
     TheDbWin(False, &dbUp, &wasdbUp);
  if (helpUp)
    TheHelpWin(False, &helpUp, &washelpUp);
  if (acsedUp)
                    TheAcsedWin(False, &acsedUp, &wasacsedUp);
  if (acsibleUp)
                    TheAcsibleWin(False, &acsibleUp, &wasacsibleUp);
}
            break;

  /*  Opcao executada quando o ponteiro entra em uma janela */

  case EnterNotify:
              if (event.xcrossing.window == users_win)
                draw_focus_frame(users_win, hicol);
```

```
            if (event.xcrossing.window == mail_win)
{
  XSetInputFocus(display, mail_win, RevertToParent, \
 CurrentTime);
                draw_focus_frame(mail_win, hicol);
}
      if (event.xcrossing.window == ed_win)
        draw_focus_frame(ed_win, hicol);
      if (event.xcrossing.window == db_win)
        draw_focus_frame(db_win, hicol);
      if (event.xcrossing.window == help_win)
        draw_focus_frame(help_win, hicol);
      if (event.xcrossing.window == acsed_win)
        draw_focus_frame(acsed_win, hicol);
      if (event.xcrossing.window == acsible_win)
        draw_focus_frame(acsible_win, hicol);
            if (event.xcrossing.window == usersnamewin || \
  event.xcrossing.window == usersareawin || \
  event.xcrossing.window == usersdiscwin || \
  event.xcrossing.window == usersstatwin)
{
  XSetInputFocus(display, event.xcrossing.window, \
 RevertToNone, CurrentTime);
  RedrawTextUsersWin(event.xcrossing.window, NOT_DIMMED);
  draw_focus_frame(users_win, hicol);
}
            if (event.xcrossing.window == mailfilewin || \
  event.xcrossing.window == mailtowin || \
  event.xcrossing.window == mailsubjwin)
{
  XSetInputFocus(display, event.xcrossing.window, \
 RevertToNone, CurrentTime);
  RedrawTextMailWin(event.xcrossing.window, NOT_DIMMED);
  draw_focus_frame(mail_win, hicol);
}
            if (event.xcrossing.window == edtopnamewin || \
  event.xcrossing.window == edtopfilewin || \
  event.xcrossing.window == edmodnamewin || \
  event.xcrossing.window == edmodfilewin || \
  event.xcrossing.window == edwinnamewin || \
```

```
   event.xcrossing.window == edwinfilewin || \
   event.xcrossing.window == edwinvarwin || \
   event.xcrossing.window == edgraphnamewin || \
   event.xcrossing.window == edgraphfilewin || \
   event.xcrossing.window == edgraphvarwin || \
   event.xcrossing.window == edtxtnamewin || \
   event.xcrossing.window == edtxtfilewin || \
   event.xcrossing.window == edtxtkeywin)
 {
   XSetInputFocus(display, event.xcrossing.window, \
  RevertToNone, CurrentTime);
   RedrawTextEditorWin(event.xcrossing.window, NOT_DIMMED);
   draw_focus_frame(ed_win, hicol);
 }
       if (event.xcrossing.window == dbtxtfilewin && \
   (dbnumelem >= 0 && dbnumelem < 4))
 {
   XSetInputFocus(display, event.xcrossing.window, \
  RevertToNone, CurrentTime);
   RedrawTextDatabaseWin(event.xcrossing.window, NOT_DIMMED);
   draw_focus_frame(db_win, hicol);
   XSelectInput(display, event.xcrossing.window, \
ExposureMask | EnterWindowMask | \
LeaveWindowMask | KeyPressMask);
   XSelectInput(display, dbfromlinkwin, ExposureMask | \
EnterWindowMask | LeaveWindowMask);
   XSelectInput(display, dbtolinkwin, ExposureMask | \
EnterWindowMask | LeaveWindowMask);
   XSelectInput(display, dbdinamewin, ExposureMask | \
EnterWindowMask | LeaveWindowMask);
   XSelectInput(display, dbdicodwin, ExposureMask | \
EnterWindowMask | LeaveWindowMask);
   XSelectInput(display, dbtpnamewin, ExposureMask | \
EnterWindowMask | LeaveWindowMask);
   XSelectInput(display, dbtpfilewin, ExposureMask | \
EnterWindowMask | LeaveWindowMask);
   XSelectInput(display, dbtpdicodwin, ExposureMask | \
EnterWindowMask | LeaveWindowMask);
 }
              if ((event.xcrossing.window == dbfromlinkwin || \
```

```
    event.xcrossing.window == dbtolinkwin) && \
    dbnumelem == 4)
  {
    XSetInputFocus(display, event.xcrossing.window, \
   RevertToNone, CurrentTime);
    RedrawTextDatabaseWin(event.xcrossing.window, NOT_DIMMED);
    draw_focus_frame(db_win, hicol);
    XSelectInput(display, event.xcrossing.window, \
 ExposureMask | EnterWindowMask | \
 LeaveWindowMask | KeyPressMask);
    XSelectInput(display, dbtxtfilewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbdinamewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbdicodwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbtpnamewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbtpfilewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbtpdicodwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
  }
if ((event.xcrossing.window == dbdinamewin || \
    event.xcrossing.window == dbdicodwin) && \
    dbnumelem == 5)
  {
    XSetInputFocus(display, event.xcrossing.window, \
   RevertToNone, CurrentTime);
    RedrawTextDatabaseWin(event.xcrossing.window, NOT_DIMMED);
    draw_focus_frame(db_win, hicol);
    XSelectInput(display, event.xcrossing.window, \
 ExposureMask | EnterWindowMask | \
 LeaveWindowMask | KeyPressMask);
    XSelectInput(display, dbtxtfilewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbfromlinkwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbtolinkwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
```

```
    XSelectInput(display, dbtpnamewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbtpfilewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbtpdicodwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
  }
if ((event.xcrossing.window == dbtpnamewin || \
    event.xcrossing.window == dbtpfilewin || \
    event.xcrossing.window == dbtpdicodwin) && \
    dbnumelem == 6)
  {
    XSetInputFocus(display, event.xcrossing.window, \
   RevertToNone, CurrentTime);
    RedrawTextDatabaseWin(event.xcrossing.window, NOT_DIMMED);
    draw_focus_frame(db_win, hicol);
    XSelectInput(display, event.xcrossing.window, \
 ExposureMask | EnterWindowMask | \
 LeaveWindowMask | KeyPressMask);
    XSelectInput(display, dbtxtfilewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbfromlinkwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbtolinkwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbdinamewin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
    XSelectInput(display, dbdicodwin, ExposureMask | \
 EnterWindowMask | LeaveWindowMask);
  }
        break;

    /* Opcao executada quando o ponteiro sai de uma janela */

    case LeaveNotify:
        if (event.xcrossing.window == users_win && \
    !(event.xcrossing.detail == NotifyInferior))
  draw_focus_frame(users_win, locol);
        if (event.xcrossing.window == mail_win && \
    !(event.xcrossing.detail == NotifyInferior))
```

```
draw_focus_frame(mail_win, locol);
      if (event.xcrossing.window == ed_win && \
  !(event.xcrossing.detail == NotifyInferior))
draw_focus_frame(ed_win, locol);
      if (event.xcrossing.window == db_win && \
  !(event.xcrossing.detail == NotifyInferior))
draw_focus_frame(db_win, locol);
      if (event.xcrossing.window == help_win && \
  !(event.xcrossing.detail == NotifyInferior))
draw_focus_frame(help_win, locol);
      if (event.xcrossing.window == acsed_win && \
  !(event.xcrossing.detail == NotifyInferior))
draw_focus_frame(acsed_win, locol);
      if (event.xcrossing.window == acsible_win && \
  !(event.xcrossing.detail == NotifyInferior))
draw_focus_frame(acsible_win, locol);
             if (event.xcrossing.window == usersnamewin || \
  event.xcrossing.window == usersareawin || \
  event.xcrossing.window == usersdiscwin || \
  event.xcrossing.window == usersstatwin)
{
  XSetInputFocus(display, users_win, RevertToNone, \
 CurrentTime);
  RedrawTextUsersWin(event.xcrossing.window, DIMMED);
  draw_focus_frame(users_win, hicol);
}
             if (event.xcrossing.window == mailfilewin || \
  event.xcrossing.window == mailtowin || \
  event.xcrossing.window == mailsubjwin)
{
  XSetInputFocus(display, mail_win, RevertToNone, \
 CurrentTime);
  RedrawTextMailWin(event.xcrossing.window, DIMMED);
  draw_focus_frame(mail_win, hicol);
}
             if (event.xcrossing.window == edtopnamewin || \
  event.xcrossing.window == edtopfilewin || \
  event.xcrossing.window == edmodnamewin || \
  event.xcrossing.window == edmodfilewin || \
  event.xcrossing.window == edwinnamewin || \
```

```
   event.xcrossing.window == edwinfilewin || \
   event.xcrossing.window == edwinvarwin || \
   event.xcrossing.window == edgraphnamewin || \
   event.xcrossing.window == edgraphfilewin || \
   event.xcrossing.window == edgraphvarwin || \
   event.xcrossing.window == edtxtnamewin || \
   event.xcrossing.window == edtxtfilewin || \
   event.xcrossing.window == edtxtkeywin)
{
   XSetInputFocus(display, ed_win, RevertToNone, \
 CurrentTime);
   RedrawTextEditorWin(event.xcrossing.window, DIMMED);
   draw_focus_frame(ed_win, hicol);
}
              if (event.xcrossing.window == dbtxtfilewin || \
   event.xcrossing.window == dbfromlinkwin || \
   event.xcrossing.window == dbtolinkwin || \
   event.xcrossing.window == dbdinamewin || \
   event.xcrossing.window == dbdicodwin || \
   event.xcrossing.window == dbtpnamewin || \
   event.xcrossing.window == dbtpfilewin || \
   event.xcrossing.window == dbtpdicodwin)
{
   XSetInputFocus(display, db_win, RevertToNone, \
 CurrentTime);
   RedrawTextDatabaseWin(event.xcrossing.window, DIMMED);
   draw_focus_frame(db_win, hicol);
}
      break;

   /* Entra nesta opcao quando alguma tecla e pressionada */

   case KeyPress:
              {
char    buffer[128];
KeySym  key_sym;
int     strlenght;

strlenght = XLookupString(&event.xkey, buffer, 128, \
   &key_sym, (XComposeStatus *) NULL);
```

```
                buffer[strlenght] = '\0';

  if (event.xkey.window == mailfilewin || \
      event.xkey.window == mailtowin || \
      event.xkey.window == mailsubjwin)
                    {
      if (key_sym != XK_Control_L && key_sym != XK_Shift_L && \
  key_sym != XK_Shift_R && key_sym != XK_Caps_Lock)
                        if (key_sym == XK_Left)
                          Mail_Keyboard(event.xkey.window, '\002');
                        else
                  if (key_sym == XK_Right)
    Mail_Keyboard(event.xkey.window, '\006');
                          else
    Mail_Keyboard(event.xkey.window, buffer[0]);
                    }
  if (event.xkey.window == usersnamewin || \
      event.xkey.window == usersareawin || \
      event.xkey.window == usersdiscwin || \
      event.xkey.window == usersstatwin)
    {
      if (key_sym != XK_Control_L && key_sym != XK_Shift_L && \
  key_sym != XK_Shift_R && key_sym != XK_Caps_Lock)
if (key_sym == XK_Left)
  Users_Keyboard(event.xkey.window, '\002');
else
  if (key_sym == XK_Right)
    Users_Keyboard(event.xkey.window, '\006');
  else
    Users_Keyboard(event.xkey.window, buffer[0]);
    }
              if (event.xkey.window == edtopnamewin || \
    event.xkey.window == edtopfilewin || \
    event.xkey.window == edmodnamewin || \
    event.xkey.window == edmodfilewin || \
    event.xkey.window == edwinnamewin || \
    event.xkey.window == edwinfilewin || \
    event.xkey.window == edwinvarwin || \
    event.xkey.window == edgraphnamewin || \
```

```
      event.xkey.window == edgraphfilewin || \
      event.xkey.window == edgraphvarwin || \
      event.xkey.window == edtxtnamewin || \
      event.xkey.window == edtxtfilewin || \
      event.xkey.window == edtxtkeywin)
    {
                      if (key_sym != XK_Control_L && key_sym != XK_Shift_L && \
  key_sym != XK_Shift_R && key_sym != XK_Caps_Lock)
        if (key_sym == XK_Left)
          Editor_Keyboard(event.xkey.window, '\002');
        else
          if (key_sym == XK_Right)
            Editor_Keyboard(event.xkey.window, '\006');
          else
            Editor_Keyboard(event.xkey.window, buffer[0]);
    }
                  if (event.xcrossing.window == dbtxtfilewin || \
      event.xcrossing.window == dbfromlinkwin || \
      event.xcrossing.window == dbtolinkwin || \
      event.xcrossing.window == dbdinamewin || \
      event.xcrossing.window == dbdicodwin || \
      event.xcrossing.window == dbtpnamewin || \
      event.xcrossing.window == dbtpfilewin || \
      event.xcrossing.window == dbtpdicodwin)
    {
                 if (key_sym != XK_Control_L && key_sym != XK_Shift_L && \
        key_sym != XK_Shift_R && key_sym != XK_Caps_Lock)
        if (key_sym == XK_Left)
          Database_Keyboard(event.xkey.window, '\002');
        else
          if (key_sym == XK_Right)
            Database_Keyboard(event.xkey.window, '\006');
          else
            Database_Keyboard(event.xkey.window, buffer[0]);
    }
        }
        break;
      }
    }
}
```

BIBLIOGRAFIA

[1] **F. Damiani, P. J. Tatsch, N. Marranghello**; *UISEE - An Interactive Software Package for Engineering Education*, Proceedings of DECONIA 89, Nsukka, Nigeria, pp 368-370, 1989.

[2] **F. Damiani, P. J. Tatsch, N. Marranghello**; *A Software for Computer-Aided Engineering Education*, National Educational Computing Conference, pp 62-64, 1990.

[3] **B. J. Cox**; *Object Oriented Programming - An Evolutionary Approach*, Addison-Wesley Publishing Company, pp 49-51, 155-161, 1987.

[4] **J. T. Berry**; *Programando em C++*, Editora McGraw-Hill Ltda., pp 72-82, 190-191, 1991.

[5] **H. F. Korth, A. Silberschatz**; *Sistemas de Bancos de Dados*, Editora McGraw-Hill Ltda., 1989.

[6] **O. Jones**; *Introduction to the X Window System*, Prentice-Hall, 1989.

[7] **A. Nye**; *X Protocol Reference Manual, Volume 0*, O'Reilly and Associates, 1990.

[8] **A. Nye**; *Xlib Programming Manual, Volume 1*, O'Reilly and Associates, 1990.

[9] **A. Nye**; *Xlib Reference Manual, Volume 2*, O'Reilly and Associates, 1988.

[10] **Sun Microsystems, Inc.**; *OPEN LOOK - Graphical User Interface Application Style Guidelines*, Addison-Wesley Publishing Company, pp 37-65, 351-371, 1990.

[11] **Sun Microsystems, Inc.**; *OPEN LOOK - Graphical User Interface Application Functional Specification*, Addison-Wesley Publishing Company, pp 16-20, 519-540, 1990.

[12] **Sun Microsystems, Inc.**; *SunView Programmer's Guide*, pp 5-13, 145-146, 1990.

[13] **Sun Microsystems, Inc.**; *Network Programming Guide*, pp 1-30, 1990.

[14] **Sun Microsystems, Inc.**; *SunOS Reference Manual*, pp 724-729, 1060-1061, 1990.

[15] **T. L. Quarles**; *The Front End to Simulator Interface*, Universidade da Califórnia - Berkeley, 1989.

[16] *SPICE3C.1 - Nutmeg Programmer's Manual*, Universidade da Califórnia - Berkeley, 1989.