UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

PÓS-GRADUACÃO EM CIÊNCIA DA COMPUTACÃO

UMA LINGUAGEM DE DEFINICÃO E MANIPULACÃO DE INTERFACES COM O USUÁRIO

por

EDSON GELLERT SCHUBERT

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

Prof. Roberto Tom Price

Orientador

Porto Alegre, janeiro de 1991

CATALOGAÇÃO NA PUBLICAÇÃO

Schubert, Edson Gellert

Uma Linguagem de Definição e Manipulação da Interface com o Usuário/Edson Gellert Schubert. Porto Alegre : CPGCC da UFRGS, 1991

225 p.

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

Orientador : Price, Roberto Tom.

Dissertação: Interfaces gráficas: Sistemas de Desenvolvimento de Interfaces com o Usuário: Gramáticas de Atributo

A DEUS,

A meus avós,

A meu irmão EDWARD,

e a meus pais, KARIN e WERNER.

AGRADECIMENTOS

Ao longo do processo de levantamento de informações, pesquisa e compilação das idéias que nortearam este trabalho, muitas amizades foram criadas e muitos contatos foram feitos, a estas pessoas meu muito obrigado pela cooperação.

Às pessoas que me deram suporte financeiro, moral e espiritual, só posso dizer obrigado por tudo.

Àqueles que auxiliaram na minha têmpera e lapidação, afirmo que o caminho foi árduo e penoso, mas recompensador.

Àqueles que me atiraram pedras ao caminho, com esperanças de me ver ao chão, somente lhes digo que venci.

Ao corpo docente do Curso de Pós-Graduação em Ciência da Computação da UFRGS, em especial ao meu orientador, pelo apoio e pelas sugestões que muito me auxiliaram na elaboração deste trabalho.

Aos funcionários da biblioteca, dos laboratórios e da secretaria pela atenção dispensada.

Aos amigos Alceu Heinke Frigeri e Marcelo Soares Pimenta pela amizade e "charlas" que tivemos à voga de futilidades e cervejas.

Aos grandes colegas e amigos Aliomar, Javan, João Paulo, Paulo Fernandes, Visintin, Remis e Alba, Benhur e Pazzini pela excelente amizade consolidada neste período.

Aos meus pais, Werner e Karin pelos ensinamentos, dedicação incansável e incentivo constante durante toda minha vida.

Aos meus avós, Alfredo e Maria Elfrida e Edmundo e Maria pela alegria e orgulho de poder dizer-lhes que grande parte do que sou e consegui é fruto de seus legados.

E principalmente a Deus, nosso Pai e Senhor, por ter me confortado, iluminado e protegido durante todos os momentos da minha caminhada.

SUMÁRIO

GLOSSÁRIO

BATCH : modo de processamento em que, uma vez iniciada a execução do programa, o usuário não tem a possibilidade de alterar/controlar seu fluxo de execução.

HARDWARE : conjunto de componentes eletrônicos combinados de forma a permitir a execução de determinadas tarefas.

IBM-PC : designam os microcomputadores baseados em processadores INTEL da família 8088. Foram inicialmente projetados e comercializados pela Internacional Business Machine (IBM), mas logo foram copiados, aperfeiçoadas e passaram a assumir uma condição de padrão internacional em matéria de computação pessoal.

ÍCONE : é um conjunto (não vazio) de símbolos gráficos que têm por função representar alguma ação ou elemento.

INTERATIVO : modo de processamento em que, uma vez iniciada a execução de um programa, o usuário tem alguma liberdade de alterar o seu fluxo de execução.

JANELA : região da tela onde o usuário interage com a aplicação.

LAYOUT : a aparência visual da interface.

MOUSE : dispositivo de apontamento de objetos no vídeo do computador

FORMULÁRIO : caixa de diálogo

PILHA : é uma lista ordenada na qual todas as inserções e retiradas são feitas numa extremidade, denominada o topo.

POINTER : seta indicando a posição do mouse na tela.

SCROLLBAR : retângulo para realizar deslocamento das informações em uma janela.

SHELL : conjunto de programas que facilitam o acesso e a utilização de determinado conjunto de recursos oferecido por um computador.

SOFTWARE : conjunto de instruções que controla e comanda o hardware de modo que, com as tarefas executadas, atinja-se determinado objetivo.

THREAD : linha ou caminho passível de ser trilhado. Pode significar uma tarefa, pois cada tarefa é formada por um conjunto de operações formando um caminho.

MULTI-THREAD: múltiplas linhas ou caminhos possíveis de serem trilhados.

TOOLKIT : conjunto de rotinas básicas para construção de interfaces gráficas.

USUÁRIO : pessoa que utiliza um sistema de computação.

WORDSTAR : editor de textos que faz largo uso de combinações de teclas para a definição dos comandos que devem ser executados.

LISTA DE ABREVIATURAS

BD	: Banco de Dados
DBMS	: Data Base Management System
SGBD	: Sistema de Gerenciamento de Banco de Dados
IU	: Interface com o Usuário
MRIU	: Modelo de Representação da Interface com o Usuário
SDIU	: Sistema de Desenvolvimento de Interfaces com o Usuário
SGIU	: Sistema de Gerenciamento de Interfaces com o Usuário
GA	: Gramáticas de Atributos
GLC	: Gramáticas Livres de Contexto
SAPPIU	: Sistemas de Apoio ao Projeto e Programação de Interfaces com o Usuário
CASE	: Computer Aided Software Engineering
MTE	: Modelo de Transição de Estados
MLE	: Modelo de Linguagem de Eventos
MGLC	: Modelo de Gramáticas Livres de Contexto
GRAEDIUS	: GRAmática Estendida para a Definição de Interfaces com os UsuárioS

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Uma interface com o usuário é composta por duas "vias" de comunicação, uma que vai do usuário até o sistema e outra que vai do sistema até o usuário. Cada uma destas "vias" possue um formalismo que define a comunicação associado.

Neste trabalho, estes formalismos são descritos com uma gramática de atributos. Esta gramática foi expandida de forma a permitir a definição dos elementos que compõe a interface do usuário, e da estrutura que irá controlar a seqüência de execução das tarefas oferecidas pelos sistemas de aplicação.

Ao longo do trabalho são discutidas algumas técnicas de descrição do formalismo de comunicação entre interface e sistema, são abordados os estilos de interação e apresentada as expansões aplicadas sobre gramáticas de atributos. Um exemplo auxilia a compreensão do uso da linguagem proposta, e um protótipo permite a validação das definições.

Palavras-Chaves : Interfaces gráficas, Sistemas de Desenvolvimento de Interfaces com o Usuário, Gramática de Atributos.

ABSTRACT

A user interface is composed by two "ways" of communication, one from the user to the system and the other linking the system to the user. Each of these "ways" has it's own mechanism.

In this work, these mechanisms are described through an attribute grammar. This grammar has been expanded to allow the definition of the structure of the interface elements and the control of the execution of the tasks that the application system implements.

Through this work, technics that describe the communication between the interface and the system, interaction styles and the extensions made on attribute grammar are discussed. An example is given to explain the use of the proposed mechanism and a prototype validates ideas discussed.

Keywords : Graphical Interfaces, User Interface Development Systems, Attribute Grammar.

1 INTRODUÇÃO

Este trabalho propõe uma linguagem de especificação e manipulação de interfaces com o usuário (IUs), de forma a facilitar as tarefas de projeto e programação destas.

Esta proposta está motivada no grande aumento que a demanda por sistemas computacionais interativos experimentou a partir do início da década de 80, com a popularização dos micro-computadores. Os sistemas que têm uma apresentação mais amigável são, usualmente, os preferidos, pois além de facilitarem o uso, dispensam grande treinamento nos conceitos computacionais e aumentam a produtividade por permitirem um aprendizado mais intuitivo. Apresentações amigáveis acabaram por se transformar em fator determinante do sucesso ou fracasso dos sistemas computacionais. As apresentações dos sistemas fazem parte da IU.

A melhoria da qualidade das IUs demanda um grande esforço nas etapas de projeto e programação dos sistemas. Em um estudo envolvendo sistemas de Inteligência Artificial (IA), os esforços referentes à IU totalizaram cerca de 40-50% do esforço total de programação [MYE 89]. O esforço de mão-de-obra envolvido com a confecção de boas IUs, traduz-

se num significativo crescimento do custo total dos sistemas computacionais[1].

Para auxiliar a confecção das IUs, busca-se a criação de sistemas que facilitem as fases de projeto e programação, de forma transparente. As primeiras tentativas nesta direção resultaram na criação de bibliotecas de funções que controlavam os periféricos dos equipamentos de forma transparente ao programador. Estas bibliotecas são conhecidas por **toolboxs** [TAK 89]. Apesar de auxiliarem as tarefas do programador de IUs, pouco contribuíram para o trabalho do projetista. Esforços neste sentido levaram a estruturas e estilos de diálogo pré-definidos. Ferramentas com estas características receberam a designação de **toolkits** [TAK 89], pois com elas o projetista somente necessita nomear as opções que nortearão o usuário final na escolha das tarefas disponíveis, enquanto que o programador somente "encaixa" as chamadas às funções de aplicação nos pontos da estrutura de diálogo em que se encontram disponíveis todas as informações inerentes às tarefas (fig. 1.1) . Um fator marcante destas ferramentas é que tanto o projetista como o programador tem a sensação de estarem trabalhando com um destes brinquedos de encaixe de pinos com formas e cores diferentes. As formas dos pinos são as chamadas às funções da aplicação, e as cores fazem as vezes dos nomes das opções.

---

[1] No início da década de 70 um problema semelhante foi o que envolvia os mecanismos de armazenamento/recuperação de informações em arquivos de dados. A solução encontrada, e hoje mundialmente utilizada, foi a criação de sistemas que se encarregassem do gerenciamento destes mecanismos de forma transparente. Estes sistemas oferecem apoio às fases de projeto das estruturas de dados utilizadas e de programação das funções de manipulação destas estruturas. Receberam a denominação de **S**istemas de **G**erenciamento de **B**ases de **D**ados (**SGBDs**) [DAT 86].

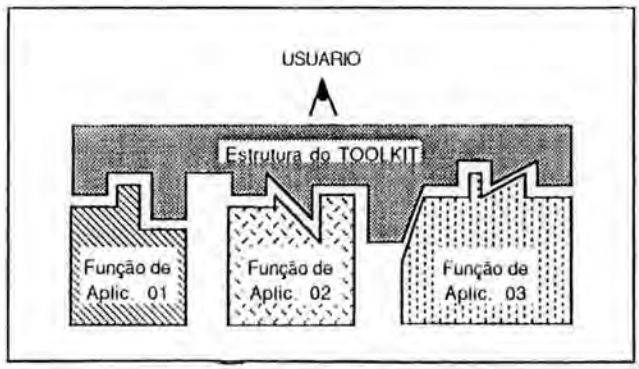

Figura 1.1 Modelo **toolkit**

O grande problema das **toolkits** é que não conferem liberdade de criação ao projetista de IUs. Os diálogos gerados são, basicamente, iguais (Vide, por exemplo, os sistemas desenvolvidos para o Macintosh). Buscando uma maior liberdade para o projetista e, simultaneamente, a possibilidade de uma documentação da IU utilizada num sistema, desenvolveram-se adaptações sobre as notações de modelos de representação de informações existentes. Com este esforço, foram propostos alguns modelos de representação da IU (MRIU), tais como o modelo de transição de estados, o de linguagem de eventos e o gramatical.

Com o objetivo de auxiliar nos trabalhos de projeto e a programação das IUs, surgiram sistemas que dão apoio ao uso dos MRIUs. As limitações que apresentam estão relacionadas com os tipos de diálogos suportados pelos MRIUs nos quais se baseiam. Por exemplo (veja-se capitulo 2), o modelo de transição de estados é adequado para IUs seqüênciais, portanto sistemas baseados nele tem facilidade de representar diálogos seqüenciais, enquanto que o modelo de linguagem de eventos é mais adequado a sistemas que implementem IUs que usem manipulação direta.

Os sistemas baseados em modelos, são divididos em dois grupos [MYE 89]. No primeiro grupo, os Sistemas de Desenvolvimento de IUs (**SDIUs**), limitam-se a apoiar as fases de projeto e programação da IU, tendo como saída trechos de programas fontes escritos em alguma linguagem (C ou PASCAL, por exemplo). As IUs obtidas como resultado do uso destes sistemas são bastante variadas, transferindo ampla liberdade de criação ao projetista. No segundo grupo, os Sistemas de Gerenciamento da Interface com o Usuário (**SGIUs**), além de darem todo o apoio às fases de projeto e programação, também permitem que o usuário final altere a IU, de forma a adaptá-la melhor às suas necessidades fisicas e psicológicas.

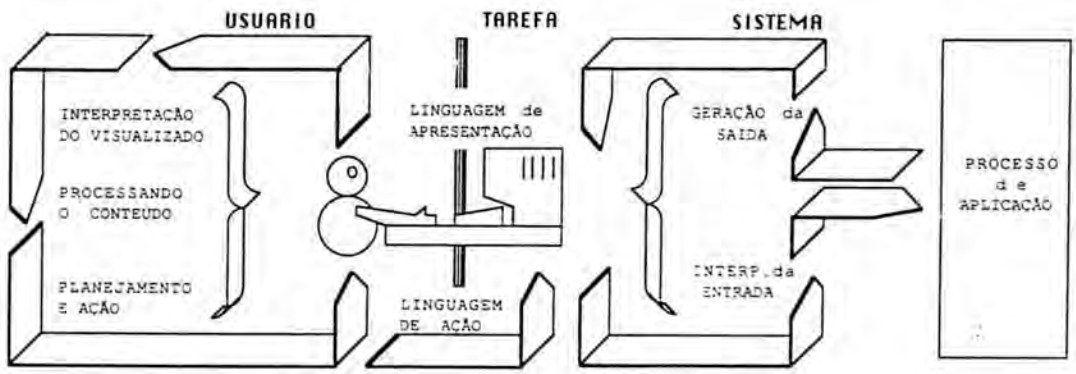

Figura 1.2 Esquema conceitual da IU de acordo com Bennett

A idéia básica, deste trabalho, é de que a interação usuário-aplicação obedece a certo esquema, seqüencial, de fatos. Bennett exprime de forma bastante clara esta seqüencialidade [BEN 86]. Para tal, divide a interação em dois macrocomponentes (fig. 1.2), a saber : a) o de ação e b) o de apresentação. O primeiro manipula o recebimento/apresentação do conjunto de tarefas que o usuário tem a disposição numa aplicação. O segundo macrocomponente é responsável por informar ao usuário o

resultado gerado pelas funções da aplicação invocadas pelas tarefas escolhidas.

O trabalho aqui proposto pode ser classificado como um SDIU. Como característica, oferece ao projetista ampla liberdade de definição da estrutura do diálogo adotado pelos sistemas, assim como permite a escolha do estilo básico de interação (menus, formulários ou linguagem de comando) que se aplicará a cada nodo (ponto de interação) da estrutura do diálogo. Por ponto de interação entende-se os nodos do grafo que definem a estrutura do diálogo, onde é necessária a intervenção do usuário para que seja determinada corretamente qual a tarefa que o usuário deseja executar. A nível de programação, permite que o grau de granularidade/acoplamento seja determinado pelo próprio programador. Estas características são afetadas pelas facilidades oferecidas pela linguagem proposta, ou seja, permite computações simples (atribuições e condições) e o uso de rotinas externas.

Partindo da premissa de que existem diálogos nos dois sentidos (usuário-aplicação e aplicação-usuário), que gramáticas são bom formalismo para a descrição de diálogos e que existem computações intermediárias aos pontos de interação de uma estrutura de diálogo, optou-se por explorar o formalismo de gramática de atributos (**GA**) estendida com algumas "seções". O objetivo destas "seções" é permitir a descrição dos dois macrocomponentes apresentados por Bennett.

No capítulo 2 encontram-se descritos três modelos de representação de IU, a saber o modelo de transição de estados, a linguagem de ação e o gramatical. No capítulo 3 são apresentados os diferentes estilos básicos de interação com o usuário, os quais são : a) menus, b) formulários, c)

linguagem de comando e d) comandos diretos. No capítulo 4 é discutida uma abordagem lingüistica da interface e relacionadas as necessidades de informação, aos níveis léxicos, sintáticos e semânticos, dos vários componentes da IU. O capítulo 5 apresenta a linguagem tema deste trabalho, enquanto seu uso com um exemplo é explicado no capítulo 6. O capítulo 7 é dedicado à discussão do protótipo que implementa um reconhecedor e tradutor da linguagem proposta em linguagem C. Por fim, no capítulo 8 encontram-se as conclusões do trabalho, assim como uma indicação das futuras tarefas a serem desenvolvidas objetivando uma complementação do trabalho.

2 MODELOS DE REPRESENTAÇÃO DE INTERFACES COM O USUÁRIO (MRIUs)

Este capítulo apresenta uma definição conceitual dos MRIUs, apontando para a sua importância na definição dos sistemas de apoio ao projeto e programação de IU (SAPPIU). Também apresenta os modelos de Transição de Estados, de Linguagem de Eventos e de Gramáticas Livres de Contexto.

2.1 Modelos

Um modelo de representação é composto por um conjunto de elementos (normalmente gráficos), que representam as informações modeladas, e um conjunto de regras que disciplina o uso destes elementos. Metodologias sempre estão baseadas em algum modelo de representação de informações.

As relações existentes entre as metodologias de projeto de IUs, que usam algum MRIU, e os SAPPIUs, é a mesma que existe entre as metodologias de análise estruturada e as ferramentas CASE que as implementam, ou seja, estas últimas implementam um conjunto de ferramentas que possibilitam o uso assistido das metodologias.

O uso de MRIUs faz-se necessário por serem os requisitos das interfaces muito instáveis e raramente bem definidos pelos usuários finais. Isto permite a preservação e organização das idéias desenvolvidas e utilizadas durante o processo de desenvolvimento [RHY 87]. O resultado desta documentação diminui os erros de projeto, pois permite sua

melhor verificação contra os requerimentos recolhidos junto ao usuário final. Outra forma de verificação, desejavelmente fornecida pelo MRIU, é a criação de uma "pré-interface" que sirva de bancada de testes de hipóteses sobre a apresentação e a operacionalidade da interface em estudo [BET 87].

Para que um MRIU possibilite uma fácil e rápida verificação da interface em estudo, é necessário que possua capacidade de capturar, com clareza e facilidade, o domínio e os procedimentos comuns das tarefas do usuário. É interessante que permita a intervenção do usuário final para que possa fazer correções ou sugestões que visem o aperfeiçoamento da interface em desenvolvimento [RHY 87].

2.2 O Modelo de Transição de Estados

O Modelo de Transição de Estados (MTE) está baseado em máquinas de estados ou em diagramas de transição/estado [GRE 86]. Sua modelagem toma por premissa que uma interface envolve a manipulação de uma seqüência de eventos de entrada (ações do usuário) [MYE 89].

Este modelo é composto por um conjunto de estados e transições [GRE 86]. Estados são cada um dos pontos geradores de ambigüidades, onde a aplicação necessita da intervenção do usuário para sua resolução. Cada transição representa uma possível solução para as ambigüidades a disposição do usuário quando o diálogo está em um determinado estado. Indicam as ações que o usuário pode realizar sobre um diálogo para, passando de um estado a outro, executar as tarefas desejadas. O estado que é o

início de todo o diálogo é chamado de **estado inicial** e da um dos estados que possuem uma tarefa da aplicação associada é chamado de **estado final.** No MTE um caminho é uma seqüência de transições que levam do estado inicial até um dos estados finais do diagrama. Uma seqüência de ações do usuário somente é aceita se existe um caminho correspondente a tal seqüência.

Este modelo possui uma notação gráfica em que os estados são representados por circulos e as transições entre os estados são representadas por arcos dirigidos (fig. 2.1). Quando o projeto de uma interface torna-se muito grande e denso, permite o uso de sub-diagramas [GRE 86]. Este subterfúgio, porém, não aumenta o poder de descrição do modelo.

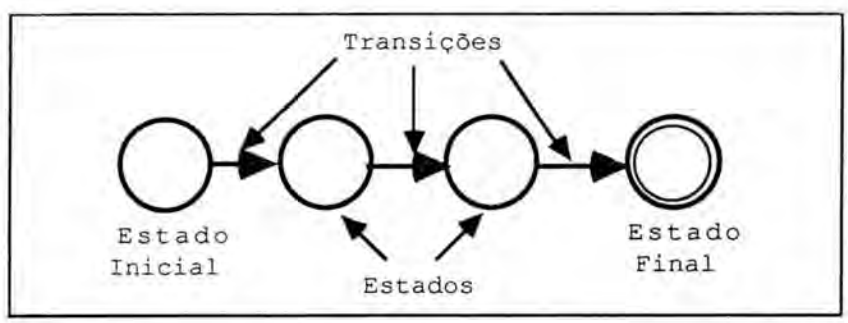

Figura 2.1 Notação usada no modelo MTE

A simples indicação dos estados e transições válidos numa interface não é suficiente para permitir uma boa documentação e compreensão do problema (um dos objetivos básicos dos MRIUs). Uma primeira variação, que visa diminuir este problema, é a que permite a adição, nos estados, de ações da aplicação (fig. 2.2), outra variação prevê nos arcos a indicação das ações da interface e nos estados as ações da aplicação (fig. 2.3).

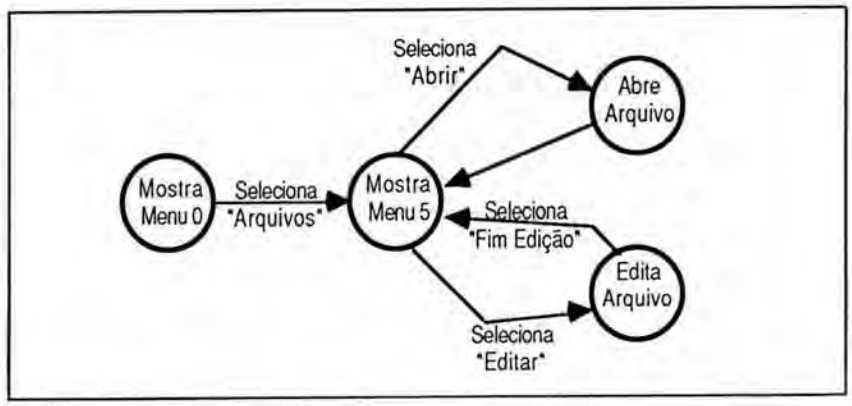

Figura 2.2 Estados indicam as ações da aplicação

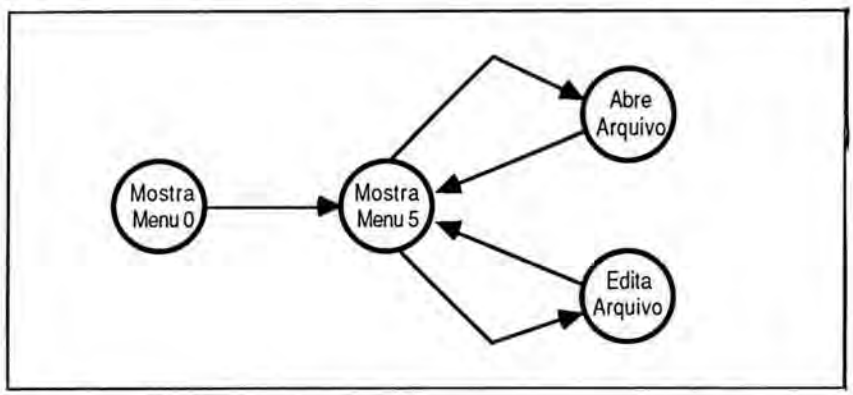

Figura 2.3 Estados indicam as ações da aplicação e as transições indicam as ações da IU

Como principais vantagens, o MTE apresenta uma representação de natureza gráfica, grande facilidade de apresentação e edição (em editores diagramáticos) [GRE 86].

Como desvantagens este modelo possue : a) dificuldade que apresenta na representação de interfaces complexas; b) quanto mais complexo um diálogo, mais densa torna-se sua representação gráfica associada [MYE 89], e mais complexa e difícil torna-se a participação do usuário na verificação da validade do projeto; c) o modelo puro (sem extensões) tem capacidade de representar um número limitado de diálogos; d) diálogos recursivos só podem ser tratados por este modelo através do uso de extensões, porém estas extensões não modelam como se processa o abandono de procedimento [GRE 86]; e e) não possue capacidade de

descrever interfaces que usem o estilo de manipulação direta (ver 3.3).

Jacob [GRE 86] propõe uma forma alterada do modelo para que possa representar interfaces onde o usuário necessite manipular vários objetos de forma concorrente (manipulação direta). Esta forma alterada prevê uma combinação entre o MTE e uma forma de linguagens de eventos (ver 2.3). Este novo modelo permite que vários diagramas estejam ativos de forma simultânea. A transferência de controle entre os diagramas se dá à maneira de co-rotinas (rotinas concorrentes, como na simulação de sistemas paralelos) [MYE 89]. Esta extensão permite a criação somente de algumas formas de manipulação direta.

Uma característica marcante deste modelo é o fato de que sua representação concentra-se exclusivamente na parte sintática da interação, ignorando os passos de definição da funcionalidade da aplicação e da especificação dos aspectos visuais do diálogo.

2.3 O Modelo de Linguagem de Eventos

No Modelo de Linguagem de Eventos (MLE) toda a interação existente entre o usuário e a interface se processa por meio de eventos. Os eventos são gerados em resposta a cada intervenção do usuário sobre os periféricos de entrada, ou pelos processos responsáveis pela manipulação destes eventos. Cada ação do usuário pode gerar um ou vários eventos. Eventos são enviados a, e tratados (processados) por, um ou vários manipuladores de eventos ("handlers"). O processamento dos eventos pode gerar outros

eventos, ou habilitar outros "handlers", ou ainda gerar chamadas às funções da aplicação [GRE 86] [MYE 89]. O projetista tem a liberdade de criar diferentes tipos de eventos, de forma que estes se adaptem às suas necessidades.

O comportamento de cada "handler" é especificado por um "template" [GRE 86]. Este "template" especifica os parâmetros, as variáveis, os eventos que manipula e ou os procedimentos (funções da aplicação) invocados, ou os eventos gerados pelo tratamento destes eventos. Cada "handler" processa, portanto, somente um conjunto limitado de eventos.

O MLE parte da premissa de que numa interação o usuário pode querer executar várias coisas em paralelo. Por isto, os "handlers" deste modelo se comportam de forma concorrente (pelo menos no nível conceitual), e não existem filas de espera, nem retardo para o processamento dos eventos. Porém como os computadores atuais realizam seus processamentos de forma seqüêncial, somente um "handler" pode estar ativo por vez, processando um evento por vez, o que gera filas de espera e retardos no processamento dos eventos.

No inicio do processamento existe sempre um "handler" que é "criado" automaticamente pelo procedimento de "carga" do sistema de aplicação. Todos os demais "handlers" são "criados" por este. A criação de outros "handlers" é comandada pelos eventos processados. É interessante notar que um "handler" está ativo, desde a sua criação até a sua destruição, não havendo momentos de desabilitação.

Um grande problema associado com o uso do MLE é que o controle de fluxo de execução não localizado dificulta a criação de código correto [GRE 86]. Esta dificuldade é parecida com a encontrada pelos programadores de linguagens não estruturadas, onde o controle do fluxo de execução torna-se tão distribuído que é praticamente inviável fazer-se qualquer controle de qualidade e exatidão das computações realizadas. É originada basicamente pelo entrelaçamento do controle.

Outros dois problemas apontados pela literatura [SHN 86] são : a) os reflexos danosos que algumas pequenas alterações numa parte do programa podem causar em outras partes (causado pelo entrelaçamento do código), e b) a dificuldade que o programador tem em entender o código quando a interface aumenta muito de tamanho. O segundo problema está basicamente associado com o controle não centralizado do fluxo de execução.

Uma das grandes vantagens deste modelo, é sua capacidade de manipulação de diálogos concorrentes (assíncronos) [GRE 86]. Estes, possibilitam o envolvimento do usuário em vários diálogos de maneira simultânea. Neste tipo de interface cada "handler" pode ser visto como um processo independente.

2.4 O Modelo de Gramáticas Livres de Contexto

O Modelo de Gramáticas Livres de Contexto (MGLC) está baseado no conceito de que toda interação entre o computador e o usuário se processa como um diálogo entre duas pessoas, e portanto pode ser descrito por uma

linguagem. O uso de gramáticas para a descrição de interfaces é uma extensão natural do uso de gramáticas para a descrição de diálogos humanos.

Uma das falhas desta abordagem, assim como das anteriores, é que linguagens são boas para definir **o que** e **o quando** se realizará cada interação, não **o como** a interação será apresentada [MYE 89]. Além de que a linguagem utilizada para a descrição da interação máquina-homem é diferente da utilizada para descrever a interação homem-máquina.

Estas caracteristicas indicam seu uso para a descrição de comandos e seus parâmetros, que caracterizem as entradas, enquanto que as saídas são, normalmente, descritas por outro formalismo [GRE 86]. O uso de dois formalismos descrevendo um mesmo objeto (interface), traz problemas de ambigüidade de descrição. O tratamento desta ambigüidade caracteriza uma das grandes desvantagens deste modelo.

A representação deste modelo tem como base uma gramática livre de contexto (GLC). Uma dificuldade associada com o uso de GLC puras neste modelo é a impossibilidade de definir as respostas geradas pelo computador. Este problema é em muito reduzido pelo uso de extensões que permitam a invocação de ações semânticas. Numa gramática assim estendida, os seus terminais são os tokens de entrada que representam as ações do usuário; as produções definem a linguagem empregada pelo usuário em suas interações com a aplicação; e as ações semânticas (da aplicação) associadas às produções são executadas quando a

produção é usada para o reconhecimento de uma entrada do usuário.

As interfaces resultantes do uso deste modelo estão baseadas em reconhecedores (parsers) [MYE 89]. Se uma gramática G descreve uma IU, então a linguagem **L(G)** descrita por esta gramática contém todas as seqüências válidas de ações do usuário. Pode-se então, a partir de G produzir um parser que reconheça **L(G)**. Os parsers assim produzidos podem ser utilizados tanto em interfaces que contenham alguma forma de conversão das opções informadas ao componente de diálogo da interface em linguagens de comando textual, quanto em interfaces baseadas em linguagens de comando textuais puras.

O problema inerente aos sistemas gerados a partir de descrições gramaticais que incorporem as extensões acima descritas é que os parsers utilizados são sensíveis ao tipo de reconhecimento empregado [GRE 86]. Se o reconhecimento for **bottom-up**, as ações somente serão executadas quando o lado direito de uma produção é reconhecido. Já se o reconhecimento for **top-down**, a execução se dá em qualquer derivação válida. Portanto faz-se necessário que o projetista da interface tenha conhecimento prévio da técnica de reconhecimento usada no parser gerado, para evitar problemas de execução indevida de alguma ação semântica.

Às vezes torna-se necessária a inclusão de produções "dummy", que funcionem como "place-holders", para contornar-se os problemas associados com a técnica de reconhecimento [GRE 86]. Uma forma de evitar-se o uso de

produções "dummy" é a ativação de ações semânticas no meio das produções.

A definição de comandos do tipo "cancele" são bastante complexas, o que torna seu uso muitas vezes praticamente inviável. Das extensões propostas, a maioria tem a ver com a recuperação de erros, com comandos do tipo "cancele" e com o controle da ordem de reconhecimento [GRE 86].

Como vantagem este modelo oferece ao projetista de IU liberdade para que se concentre nas informações passadas e recebidas do usuário, despreocupando-se da seqüência dos eventos (tokens de entrada). Já como desvantagem tem que as IUs geradas por este modelo são fundamentalmente baseadas no estilo de linguagens de comandos, bastante complicado para usuários ocasionais (ver 3.2). O uso de GLCs puras neste modelo acarretam problemas ligados com a definição da funcionalidade e com a especificação dos aspectos visuais e metafóricos do diálogo associado, além de concentrar-se exclusivamente na parte sintática da IU [OLS 86].

3 METÁFORAS E ESTILOS DE INTERAÇÃO

Neste capítulo primeiramente é descrita a característica que divide em grupos os diferentes estilos de interação utilizadas nas IUs. Neste trabalho, técnicas de interação são interpretadas como as combinações de diferentes estilos de interação para a obtenção de uma IU. Diz-se "combinação", por ser muito difícil que uma IU use somente um estilo de interação. As seções 3.2 e 3.3 discutem detalhadamente cada um destes estilos.

Os diferentes estilos de interação existentes, são os métodos disponíveis às aplicações para que realizem a comunicação com seus usuários (i.e. menus, formulários, linguagens de comando e comando direto). O uso puro ou combinado destes estilos é que determina a efetividade e a consistência de uma IU [RHY 87].

Diferentes combinações de estilos de interação, participando de uma IU, fazem com que esta seja mais apropriada aos diferentes tipos de usuários. Assim, o estilo mais indicado para usuários **sofisticados** (ou experientes) é a linguagem de comando, enquanto que para usuários **ocasionais** (ou inexperientes) são mais indicadas as IUs que usem uma combinação de menus, formulários e comando direto. Fisher em [FIS 88] entende por usuário sofisticado aquele que, conhecendo a aplicação, seleciona o próximo comando e operação que a aplicação deverá executar. Suas principais características são : a) é freqüente usuário da aplicação; b) tem longas utilizações da aplicação; e c) é usuário familiarizado com o domínio da aplicação. Já um usuário ocasional é aquele que não possue grande intimidade com o computador, ou que não utiliza determinada aplicação com fre-

qüência tal que lhe garanta absoluto controle sobre sua utilização. As principais características deste grupo de usuários são : a) as utilizações das aplicações são rápidas; b) as utilizações são esporádicas; e c) o uso das aplicações é ocasional.

Devido as diferenças existentes entre os usuários sofisticados e os ocasionais, é aconselhável que as IUs de aplicações que sirvam a estes dois grupos sejam obtidas a partir da combinação de vários estilos de interação [RHY 86]. Esta combinação fornece uma melhor adequação das IUs geradas aos diferentes usuários [SHN 86]. A mudança de estilo de interação é usada quando uma adequada visualização das ações ativas não é encontrada com o estilo em uso.

3.1 Metáforas

A divisão dos estilos de interação em diferentes categorias, está baseada na identificação das metáforas básicas do diálogo homem-máquina [HAR 89]. Hutchins et ali [HUT 86] cita a existência de pelo menos duas metáforas de descrição das interações, a saber : a) a **conversacional**, caracterizada pelos diálogos seqüênciais ou síncronos, e b) a **ideal** caracterizada pelos diálogos assíncronos, ou concorrentes. Sibert também divide os estilos em categorias equivalentes às citadas por Hutchins [SIB 86]. Somente o nome muda, em vez de conversacional usa **lingüistico** e em vez de ideal usa **espacial**, mas a idéia que norteou a divisão é a mesma.

3.1.1 Metáfora Conversacional

Na metáfora conversacional, o usuário descreve o que deseja fazer, tipicamente, pelo uso de alguma espécie de linguagem de comandos [HAR 89]; ou seja, o usuário pressupõe que está dialogando, conversando, com a aplicação e dizendo-lhe o que deseja que seja feito. Esta metáfora "vê" a interface como um diálogo entre o usuário e a aplicação [SIB 86].

Nos diálogos, seqüênciais ou síncronos, desta metáfora as passagens de uma parte do diálogo para outra ocorrem de forma previsível, permitindo ao projetista de IU e ao usuário final uma visualização do comportamento de uma seqüência lógica específica de ações de entrada [HAR 89]. A metáfora conversacional fornece um valioso "framework" para a focalização nos conteúdos que ocorrem nos níveis semânticos, sintáticos e léxicos do diálogo, encorajando o projetista a visualizar cada um destes níveis isoladamente [SIB 86] [RHY 87].

Os diálogos classificados nesta metáfora incluem os que possuem interações do tipo pergunta-resposta, cadeias de comandos e navegação por redes de menus. Como limitações pode-se citar sua dificuldade para a descrição de interfaces de comando direto (ver 3.3).

Além de classificados, os diálogos pertencentes a esta metáfora podem ser organizados [BET 87] em :

- **planos**, são aqueles em que não existe uma estrutura imposta aos comandos, ou seja, todos são acessíveis em qualquer ponto do diálogo;

- **hierárquicos**, são aqueles em que em cada ponto da estrutura de diálogo, somente os comandos daquele ponto podem ser utilizados. Qualquer outro comando exigirá a navegação pela estrutura;

- **hierárquicos com desvios limitados**, são os diálogos hierárquicos que possuem mecanismos de atalhos, que possibilitam o uso de comandos de outros pontos sem a necessidade de navegação pela estrutura.

3.1.2 Metáfora Ideal

Na metáfora ideal, a descrição do diálogo tenta refletir melhor a realidade da interação, ou seja, visa permitir que o usuário "mostre" para a aplicação o que ele quer que seja feito, indicando e manipulando representações (normalmente gráficas) dos objetos [HAR 89].

A inexistência de mecanismos que controlem/manipulem as seqüências sintáticas dos eventos permite a fácil descrição de interfaces que simulam a manipulação direta de objetos no computador. Estes mecanismos estão bem definidos na metáfora conversacional.

Os diálogos desta metáfora são implementados pelo estilo de interação denominado comando direto [HAR 89]. Nesta metáfora, várias **threads** de diálogo estão disponíveis num mesmo momento. Geralmente associados a diálogos que possuem comando direto, encontram-se os conceitos de **multithread** [BET 87] [HAR 89]. Diálogos multithread tem um con-

ceito orientado a tarefa, caracterizado pelos múltiplos caminhos de tarefas disponiveis ao usuário final em qualquer momento da interação [HAR 89].

Os diálogos desta metáfora são ditos assincronos por ser a seqüência de ações do usuário para uma tarefa, independente das seqüências das outras tarefas, ou seja, em quase qualquer momento da interação, o usuário final pode iniciar, no meio de outra tarefa, a realização de outra tarefa e, depois, retornar a execução interrompida da primeira.

3.2 Estilos Seqüênciais

O conjunto de estilos classificados como seqüências são o menu (ou cardápio), o formulário e a linguagem de comando. Como caracteristica comum tem-se o fato de que o usuário escolhe o que deseja que seja executado (que tarefa) dentre um conjunto de tarefas disponíveis. No caso de menus e formulários estes conjuntos são apresentados explicitamente, enquanto que nas linguagens de comando este conjunto deve ser memorizado pelo usuário.

3.2.1 Menus

Neste estilo o usuário primeiro lê uma lista de itens (como num cardápio), seleciona o mais apropriado, aplica a sintaxe que define a sua seleção, confirma a sua escolha, inicia a ação e observa o efeito. Se a terminologia e significado dos nomes das itens apresentados é compreensivel e familiar, no contexto do dominio da aplicação,

então o usuário necessitará de pouco treinamento ou memorização e necessitará selecionar poucos itens para realizar suas tarefas. Um dos grandes benefícios deste estilo é a facilidade, encontrada pelo usuário final, para estruturar o que fazer, visto que somente algumas opções são apresentadas por vez.

A facilidade de estruturação de uso de uma aplicação guia o usuário ocasional na resolução de seus problemas. Com um projeto adequado e com alta velocidade de resposta a uma seleção e apresentação do conjunto de itens, este estilo pode ser adequado a usuários sofisticados. Uma das formas de obter-se mais facilmente esta adequação, é pelo uso de mecanismos que permitam ao usuário uma rápida movimentação através dos conjuntos de itens apresentados. Alguns destes mecanismos são nomes únicos de itens, o uso de teclas que levam diretamente à tarefa desejada, ou macros de repetição das intervenções do usuário sobre os dispositivos de entrada com facilidades que permitam a solicitação de informações ao usuário (funcionam como uma linguagem de programação da aplicação).

Pelo fato de apresentar ao usuário o conjunto de itens que são passiveis de seleção, este estilo diminui em muito a possibilidade de erros léxicos.

Uma variante bastante atraente de menus é o dito **menu embutido** [SHN 86]. Tem como principal característica a diluição dos nomes dos itens de seleção dentro de um texto (fig. 3.1). Como esta é uma variação que somente altera a forma de apresentação dos menus, seu uso pode ser feito em qualquer local onde estes sejam aplicáveis.

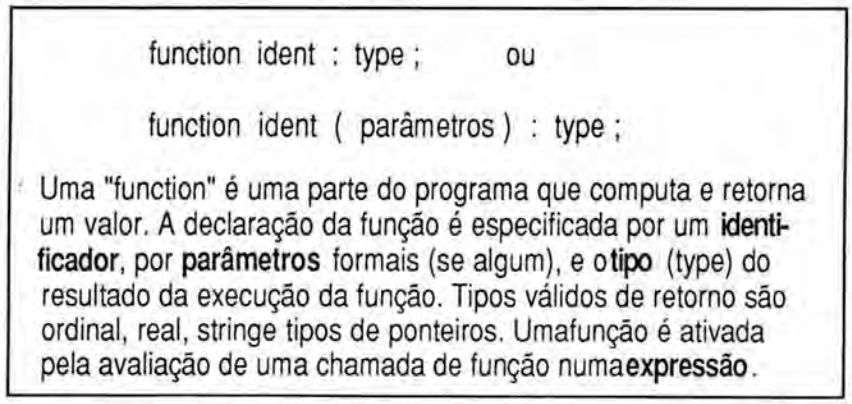

Figura 3.1 Exemplo de menu embutido

Os menus podem ser organizados segundo as seguintes variações :

- menus simples;
- de seqüência linear;
- com estrutura de árvore;
- com redes ciclicas; e
- com redes acíclicas.

3.2.1.1 Menus Simples

Os menus simples podem ser apresentados de maneira convencional, ou de forma a "aparecer" em resposta, por exemplo, a um **click** de **mouse**. Menus com esta forma de apresentação são chamados de **menus pop_up**. Os menus simples ainda são sub-organizados da seguinte forma :

- **binários** : são menus com escolhas do tipo SIM/NÃO ou Verdadeiro/Falso (fig. 3.2);

Figura 3.2 Menu binário

- **itens múltiplos** : é o tipo mais comum de menu, é onde o usuário escolhe somente um dos itens apresentados (fig. 3.3);

Figura 3.3 Menu múltiplo

- **estendidos** : são os menus onde a quantidade de itens que devem ser apresentados é maior que o espaço disponivel; nestes o usuário também só escolhe um item (fig. 3.4);

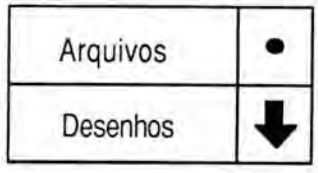

Figura 3.4 Menu estendido

- **permanentes** : a característica destes menus é que os itens estão permanentemente disponíveis para seleção (fig. 3.5);

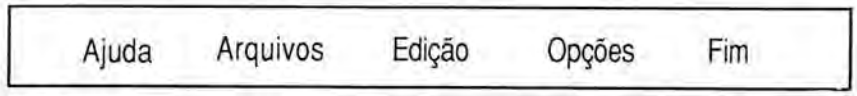

Figura 3.5 Menu permanente

- **múltiplas seleções** : a apresentação visual destes é idêntica à dos menus simples, porém permitem que vários itens sejam selecionados numa mesma interação (fig. 3.6);

Figura 3.6 Menu de múltiplas seleções

3.2.1.2 Menus de Seqüência Linear

Os menus de seqüência linear são usados quando uma determinada seqüência de itens é apresentada, de forma independente das seleções já realizadas. São altamente indicados para a recepção de parâmetros de comandos. Além disto, guiam com facilidade o usuário nos processos complexos de decisão, pois apresentam uma decisão por vez (fig. 3.7).

Figura 3.7 Menu de seqüência linear

3.2.1.3 Menus Estruturados em Árvores

Nos menus estruturados em árvores, os itens são agrupados, por semelhança, em categorias. Na fase de projeto, mesmo que os agrupamentos tragam confusões, através da intervenção do usuário é possivel chegar-se a bons resultados. Para se ter uma idéia da potencialidade deste tipo de menus, uma estrutura que contenha menus de 8 itens e com 4 niveis de profundidade pode armazenar 4.096 identificadores de tarefas nas suas folhas. A inconveniência está na vulnerabilidade a que os sistemas se expõe, devido a uma má organização das tarefas ao longo dos menus.

Nos menus estruturados em árvores, estudos tem apontado como números ideais para a relação entre a largura e a profundidade, uma distribuição de 4 a 8 itens por menu com 2 níveis de profundidade [SHN 86]. Quanto ao agrupamento semântico adotado nas estruturas, existem algumas regras que facilitam esta atividade :

- os grupos devem possuir itens logicamente similares;
- criar grupos que cubram todas as possibilidades de seleção;

- certificar-se que não existem itens redundantes;
- usar terminologia familiar ao domínio da aplicação, mas certifique-se que os itens são distinguíveis uns dos outros.

Uma maneira de facilitar o uso de menus quando o número de itens torna-se muito grande, é usar de mapas detalhados das estruturas, com indicação da posição do diálogo dentro da estrutura.

Menus com redes ciclicas ou aciclicas são muitas vezes necessários para permitir que de diferentes pontos do diálogo o usuário possa chegar a um mesmo ponto. O inconveniente é que o uso de ciclos aumenta muito a possibilidade de que o usuário venha a se perder na estrutura do diálogo. Esta desorientação é devido ao fato de que modelos mentais são difíceis de serem formados em estruturas deste tipo.

Diferentes formas de organização (alfabética ou semântica por exemplo) dos itens de um menu não trazem diferenças significativas nas obtenções (execução) das tarefas. Numa experiência que solicitava um item de uma lista permanentemente exposta de 18 itens, a ordenação alfabética e funcional apresentaram pouca diferença nos seus tempos de busca, que foram muito elevados para ordenações aleatórias. São preferíveis as organizações que tragam um menor caminho de busca para que uma tarefa seja completamente especificada [SHN 86].

Quando há a necessidade de ordenação da apresentação dos itens, algumas das organizações naturais tipicamente disponíveis ao projetista são as ordenações cronoló-

gicas, as numéricas e as que envolvem propriedades físicas. Não existindo organizações naturais, o projetista pode lançar mão de ordenações tão variadas quanto seqüenciamento alfabético, agrupamento por inter-relação de itens, por freqüência de utilização e por importância dos itens.

No projeto da apresentação física dos menus, devem ser observados :

- **títulos** : ajudam na orientação (indicam o local) do usuário na navegação pela estrutura do diálogo;
- **itens** : usar terminologia concisa, consistente, não ambigua e familiar dando destaque às palavras chaves;
- **mensagens de erro** : devem aparecer em local consistente;
- **informações de *status*** : é um grande aliado contra a desorientação quando indica onde, na estrutura, o menu corrente se encontra;
- **consistência entre menus** : itens com mesmas funções em local e nomes idênticos.

3.2.2 Formulários

Este estilo é assim denominado por simular um formulário de papel e permitir a informação de um grande número de dados (campos ou parâmetros). Na sua utilização os usuários vêem a apresentação dos campos relevantes, movem o cursor através deles e informam os valores das informações onde se faça necessário.

Os rótulos usados devem ser significativos, além de ser necessário que o usuário conheça os domínios válidos para os dados informados, para que seja capaz de responder as mensagens de erro. O treinamento associado deve facilitar o entendimento destes elementos.

Em uma comparação com linguagens de comando, em uma operação de atualização de uma base de dados, formulários apresentaram uma significativa diminuição do tempo gasto, além de ser o estilo preferido, para esta tarefa, de 11 dos 12 participantes desta comparação.

No projeto de apresentação do formulário, as seguintes considerações são interessantes de serem observadas :

- **títulos** : deve identificar o tópico em questão, sem considerar as terminologias computacionais;
- **instruções compreensíveis** : as tarefas devem ser descritas com terminologia familiar. Caso haja a necessidade de mais informações, devem ser utilizadas telas de ajuda auxiliares;
- **agrupamento e seqüenciamento** : campos relacionados devem ser adjacentes, alinhados e possuir seus grupos separados de forma clara (talvez por brancos);
- **disposição agradável** : uma distribuição uniforme dos rótulos e campos na área do formulário é preferível ao agrupamento. Alinhamentos criam uma sensação de ordem e facilitam a compreensão;

- **rótulos familiares** : o uso de termos comuns para a denominação dos campos facilita seu uso e a compreensão para o usuário. O rótulo "nome" é preferível à "XYZ" para designar um campo que deve receber o nome do funcionário;

- **terminologia e abreviaturas** : preparar uma lista de termos e abreviaturas aceitáveis e compreensíveis para serem discutidas e adotadas pelos usuários. O uso desta lista deve ser feita de forma criteriosa dentro do sistema, e se adições forem necessárias, somente efetivá-las após criteriosa diligência junto ao usuário;

- **determinação visual do tamanho e limites dos campos** : a indicação do número de caracteres ajuda ao usuário, entre outras coisas, a gerenciar as abreviaturas;

- **movimentos convenientes do cursor** : mecanismos simples e visiveis são necessários;

- **máscaras de edição** : o uso de máscaras (formatos) de edição para campos numéricos, endereços, etc, facilita o entendimento do usuário;

- **correção de caracteres** : mecanismos que implementem retrocessos e sobre-escrita são interessantes, pois facilitam o trabalho de digitação/correção dos valores dos campos solicitados;

- **mensagens para entradas incorretas** : quanto mais claras e objetivas forem as mensagens melhor;

- **campos opcionais** : as indicações dos campos opcionais são necessárias para o norteamento do usuário. Os campos opcionais devem, sempre que possível, seguir os obrigatórios, nunca antecedê-los nos formulários, pois facilitam ao usuário a identificação e digitação dos valores dos campos obrigatórios;

- **mensagens explanatórias por campo** : mensagens que indiquem que tipo de informação se espera no campo sob o cursor devem aparecer sempre que possível;

- **sinal de fim de preenchimento** : fornecer ao usuário um comando, campo ou outro mecanismo que lhe permita indicar o que deve ser feito quando o formulário estiver preenchido, ou quando desejar abandonar a operação.

Existem ainda algumas importantes considerações sobre os estilos utilizados para o recebimento das informações fornecidas pelos usuários finais. Algumas das regras discutidas por Smith e Mosier [SHN 86] são apresentadas a seguir :

- **consistência das transações** : deve-se buscar manter a semelhança entre as entradas de dados, sob qualquer circunstância;

- **minimizar as ações de entrada do usuário** : a diminuição do número de ações de entrada levam o usuário a obter uma maior produtividade com um menor índice de erros. Constantes trocas de periféricos de entrada (teclado, **mouse**, **lightpen**, etc), além de aumentar o desconforto de uso, aumenta as chances de erro e diminuem a eficiência do usuário. É recomendado o uso de mecanismos de cópia quando existirem redundâncias entre entradas;

- **reduzir a necessidade de memorização** : a memorização de longas listas de códigos e sintaxes de comandos complexos são desaconselhadas, pois além de incômodas demandam um longo período de treinamento;

- **compatibilizar as entradas com as saídas** : as máscaras utilizadas para a apresentação das informações de saída devem ser as mesmas usadas para a entrada de seus valores;

- **flexibilidade no controle das entradas** : usuários sofisticados gostam de controlar a seqüência das informações entradas, pois dá-lhes a sensação de poder dominar a aplicação. Mas um cuidado especial deve ser tomado, pois o aumento de flexibilidade vai contra os princípios de consistência.

3.2.3 Linguagem de Comandos

O estilo de linguagem de comandos está baseado num ciclo de **comando-execução-comando-execução-...** Neste ciclo o usuário especifica o comando que o sistema executa imediatamente, depois o usuário especifica outro comando e assim sucessivamente [MOR 81].

Os sistemas operacionais foram os primeiros sistemas que apresentaram linguagens de especificação das tarefas, mas distinguem-se dos estilos pelo impacto que produzem sobre os periféricos. Assim como menus, comandos podem também ser bastante simples ou até possuir uma sintaxe complexa, mas preferencialmente devem possuir uma estrutura hierárquica, ou pelo menos permitir concatenações que formem as variações necessárias. Podem executar poucas ou dezenas de operações.

Linguagens se diferenciam de menus pelo fato de que os usuários de linguagens de comando devem relembrar as notações dos comandos e iniciar as ações. Isto é necessário porque em menus o usuário somente necessita reconhecer e selecionar os itens que especificam as tarefas que deseja executar, enquanto que em linguagens de comando os usuários são estimulados a juntar memorização de comandos com sua digitação, o que aumenta muito a possibilidade de erros.

Este estilo tem associado um elevado volume de erros, imprescindíveis treinamentos e uma baixa taxa de memorização. A necessidade de mensagens de erro claras e ajuda **on-line** é óbvia, mas estas são muito difíceis de serem projetadas e oferecidas, tendo-se em vista a variedade de comandos possíveis, além da complexidade de mapeamento das tarefas sobre os conceitos computacionais e sintáticos oferecidos. Porém uma vez que o usuário tenha aprendido a sua sintaxe, pode rapidamente formular comandos complexos, sem a necessidade de ler os rótulos de orientação (**prompts**).

Apesar destes inconvenientes, quando o número de tarefas é reduzido e existe um comando associado a cada

uma, podem ser produzidos sistemas simples de aprender e usar.

O uso de **prompts** é recomendado, mas devem ser reduzidos. **Prompts** que contenham um rápido resumo dos comandos disponíveis (como no **WordStar**, por exemplo), além de facilitar o uso da aplicação para usuários inexperientes, não atrapalha os usuários experientes, uma vez que estes não necessitam ler os prompts dos comandos por já os terem memorizado. Cuidado especial deve ser tomado no projeto destes resumos para que não possibilitem a interpretação do estilo como sendo de menu.

O sistema de controle deste estilo deve emitir avisos de realimentação (**feedback**), informando da aceitação dos comandos digitados, assim como emite mensagens de erro para comandos desconhecidos.

Tipicamente, um comando é formado por um verbo, por um objeto e pelos seus argumentos (parâmetros). Os argumentos podem ser do verbo ou do objeto em questão. O uso de abreviaturas é desejável, mas seu uso está atrelado ao prévio conhecimento e domínio das técnicas de abreviatura adotadas. Apesar de serem grandes geradoras de erros, as abreviaturas são muito apreciadas pelos usuários sofisticados [SHN 86].

Opções em comandos permitem o endereçamento de casos especiais de tarefas, mas quando o número de opções e argumentos aumenta muito, aumentam também o número de erros; apesar disto, é uma característica bastante apreciada por usuários sofisticados (talvez pela possibilidade de que

com seu domínio, assume-se um **status** de **guru** do sistema !). Especial atenção deve ser dada a uma consistente ordenação dos argumentos, pois assim evitam-se confusões e erros desnecessários. O uso de rótulos para os argumentos ajuda na sua identificação e pode ser bastante útil para alguns usuários.

Na organização dos comandos, uma estrutura de árvore pode ser adotada. Esta árvore pode ter as regras de construção adotadas para menus, ou seja, por exemplo, o primeiro nível pode ser o das **ações**, o segundo o dos **objetos** das ações e o terceiro pode ser o **destino** do objeto. No estudo dos nomes dos comandos, uma orientação a palavras-chaves do domínio da aplicação traz melhores resultados finais do que uma orientação a símbolos [SHN 86].

3.3 Estilos Assíncronos

O estilo de comando direto é utilizado na implementação de diálogos assíncronas.

3.3.1 Comando Direto

Uma das principais características deste estilo é a alta dinamicidade que confere à interface, pois dá ao usuário uma realimentação quase que instantânea das suas ações [SHN 86]. Isto só é possível graças à estreita inter-relação existente entre a aplicação e o componente de interface, possibilitando uma rápida resposta semântica às ações do usuário.

O termo comando direto é melhor exemplificado quando a algumas ferramentas industriais robotizadas. Nestas ferramentas, a realização de uma tarefa é feita pela repetição dos movimentos realizados pelo usuário. O uso desta técnica na área computacional implementa uma simulação do mundo real, onde os objetos da aplicação possuem uma representação gráfica (visual) e podem ser manipulados pelo usuário final como se fossem reais.

A manipulação de representações gráficas dos objetos é mais natural e próxima das capacidades humanas. Ações e sensações visuais vieram bem antes da evolução lingüística humana. A criação de representações gráficas do universo de ações, simplifica em muito as tarefas do usuário, quando se permite a manipulação direta dos objetos de interesse. O apontamento das representações gráficas dos objetos e ações, permite que : a) o usuário defina com rapidez suas tarefas, e b) observe rapidamente os resultados destas.

Quando o número de objetos é reduzido e sobre eles aplicam-se tarefas simples, é fácil e simples visualizar e projetar um diálogo com comando direto, mas em aplicações mais complexas o projeto torna-se difícil. Para exemplificar, imaginar a representação gráfica de um arquivo de dados (fig. 3.8) é relativamente fácil, mas imaginar uma representação única, clara e concisa para um grupo de arquivos que tenham em comum o valor de um de seus campos é algo bastante complexo.

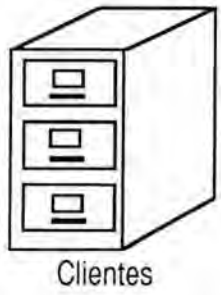

Figura 3.8 Representação gráfica de um arquivo

Para facilitar o aprendizado, sem prejuízo do desempenho nem do conforto de uso dos usuários sofisticados, são aconselhadas analogias gráficas familiares, enquanto que as representações mais abstratas são mais indicadas para o uso regular. Uma forma de auxiliar o treinamento é fornecer uma descrição explícita do modelo, das premissas e das limitações do sistema.

De acordo com MacDonald [SHN 86], a programação visual[1] das IUs (feita com algum editor de IUs) é a solução para as deficiências dos programadores de aplicação. Crê que a programação visual, além de aumentar a velocidade de construção dos sistemas, permite aos usuários finais a geração e modificação dos sistemas de aplicação para adaptá-los às suas necessidades.

Para que as conclusões de MacDonald possam ser aceitas, é necessária a existência de bibliotecas com procedimentos para a manipulação de vídeo, entrada de dados e até de computações mais complexas. Caso contrário, a própria programação dos aspectos visuais envolveria um esforço considerável, além de consumir um tempo significativo

[1] A programação visual preve a utilização de editores de recursos na definição dos aspectos visuais dos componentes utilizados em uma IU.

frente ao tempo gasto para a confecção de todo o sistema [MYE 89].

No projeto de diálogos que usem comando direto, alguns princípios devem ser observados.

- **representação contínua** : uma contínua representação dos objetos (como nas ações de movimentação dos ícones de arquivos) e ações de interesse, facilita ao usuário o entendimento da definição das tarefas;

- **ações físicas ou botões rotulados** : colocar uma representação de um arquivo sobre a representação de uma lixeira e ver a primeira representação sumir, é mais significativa e fácil de aprender que sintaxes complexas;

- **operações de reversão** : permitir reversões rápidas e incrementais onde o impacto sobre o objeto de interesse seja rapidamente visualizado.

Observando-se os princípios acima descritos, as seguintes vantagens são observadas :

- usuários ocasionais podem aprender os princípios básicos rapidamente, normalmente uma demonstração feita por um usuário mais experiente é suficiente;

- usuários experientes podem executar rapidamente uma ampla gama de tarefas, mesmo com a adição de novas funções e representações;

- usuários ocasionais retém conceitos operacionais com mais facilidade;

- raramente são necessárias mensagens de erro;
- o retorno semântico das ações do usuário são imediatas;
- usuários sofisticados tem menos problemas com a expectativa de respostas, por serem de fácil compreensão, pela transparência do sistema e pela facilidade de reversão das ações;
- a auto-confiança dos usuários cresce; por serem eles os ativadores das ações, sentem que estão no controle da aplicação e as respostas do sistema são previsíveis.

Como desvantagens o estilo de comando direto apresenta :

- uma informação incorreta, ou uma representação pouco precisa, podem criar confusão aos usuários;
- os usuários devem aprender o significado dos componentes das representações gráficas adotadas;
- as representações gráficas podem permitir interpretações duplas dos seus reais significados;
- o espaço ocupado pelas representações gráficas pode ser muito grande para a tela;
- o tempo que os usuários sofisticados levam para movimentar o apontador e definir as tarefas, pode ser maior que o tempo de digitação dos correspondentes comandos.

4 UMA VISÃO LINGÜÍSTICA DAS IUs

Neste trabalho é adotada a idéia apresentada por Coutaz [COU 85] e Hudson e King [HUD 86] de que a IU, do ponto de vista do usuário, implementa uma linguagem que lhe permite "controlar" a aplicação. Este ponto de vista é explorado neste capitulo, quando apresenta a IU sob a ótica lingüistica, ou seja, quando apresenta a IU dividida de acordo com seus componentes léxicos, sintáticos e semânticos. Esta divisão é enfocada na seção 4.1. Na seção 4.2 são descritos as informações necessárias aos estilos de interação apresentados no capítulo 3. Nas seções 4.3 e 4.4 são descritos as necessidades de informação para descrição dos demais componentes da IU.

4.1 A Divisão da Interface

Visualizar as IUs como sendo implementações de linguagens que permitem ao usuário final "controlar" a aplicação, traz como vantagem imediata a possibilidade de traçar-se uma analogia com linguagens de programação. Nesta visão, as linguagens de programação executam o controle da aplicação.

As IUs, assim como as linguagens de programação, também possuem componentes léxicos, sintáticos e semânticos [COU 85]. Esta estruturação permite que cada um destes componentes seja analisado separadamente. Esta análise fracionada facilita os processos de reconhecimento, detecção de erros e compilação ou interpretação. Neste trabalho, as definições que determinam cada um destes componentes são :

- **léxicos :** nas análises dos componentes léxicos das linguagens de programação, a cadeia de caracteres que forma um programa fonte é lida da esquerda para a direita (lexograficamente) e agrupada em **tokens**. Tokens são seqüências de caracteres com significado no conjunto [AHO 86].

- **sintáticos :** na análise dos componentes sintáticos das linguagens de programação, os **tokens** são agrupados em **sentenças gramaticais** que possuem significado no conjunto [AHO 86].

- **semânticos :** na análise dos componentes semânticos das linguagens de programação, são feitas determinadas verificações para certificar-se que as **sentenças gramaticais** tem significado quando juntapostas [AHO 86].

As sentenças gramaticais, descritas pelos componentes sintáticos, quando juntapostas, formam uma estrutura em árvore chamada *árvore de sintaxe abstrata* [FAV 87]. Nesta árvore estão descritas todas as possíveis sentenças da linguagem. Os componentes que descrevem esta árvore formam a sintaxe abstrata.

Quando se analisa linguagem de programação como o LISP [OAK 86], nota-se que as aplicações com ela escritas

são compostas por chamadas a funções[1]. Os programas escritos nesta linguagem nada mais são do que um conjunto de chamadas a funções.

Quando se analisa uma aplicação, uma divisão modular permite a identificação de pelo menos dois conjuntos de componentes : um responsável pelo processamento das informações manipuladas pela aplicação e outro responsável pela apresentação das informações ao usuário (também podem ser identificas como funções da IU). O primeiro grupo, quando estruturado numa visão espacial, permite a identificação das funções que controlam a execução de uma determinada tarefa, além de definir o formato de interfaceamento com as funções do segundo grupo (figura 4.1).

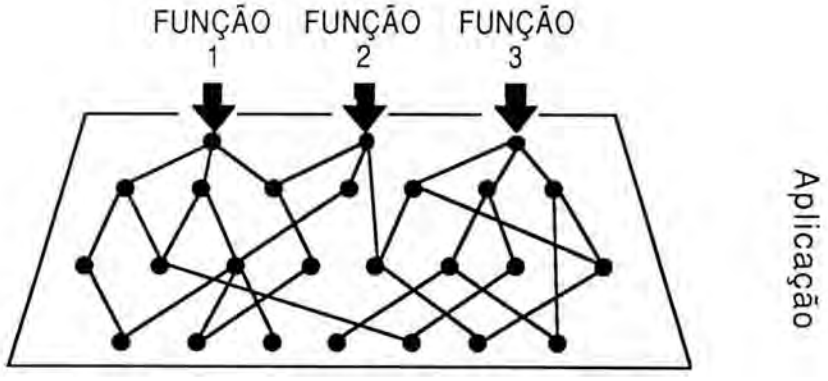

Figura 4.1 Aspecto espacial de uma aplicação

Sob a ótica lingüistica, cada sentença gramatical da IU define uma seqüências de ações que o usuário deve

[1] A linguagem PROLOG [BOR 86] também pode ser analisada desta maneira, desde que se interprete que os predicados são equivalentes a funções.

realizar para especificar/executar uma tarefa. Cada tarefa é executada por uma chamada à correspondente função de definição da tarefa na aplicação. Esta correspondência (linguagens-IU) permite dizer que uma estrutura de diálogo é totalmente mapeada numa árvore de sintaxe abstrata, e que as chamadas às funções de aplicação devem estar associadas aos nodos terminais desta árvore.

A IU é quem oferece ao usuário a capacidade de especificar e executar as tarefas disponíveis num sistema. Para isto, a IU conta com o auxílio de um conjunto de elementos de interação. Por elemento de interação entende-se cada um dos menus, formulários, etc - representantes dos estilos de interação - usados para apresentar/solicitar ações ao usuário. Uma IU pode ser composta por elementos de interação pertencentes a vários estilos de interação. A qualidade da IU de um sistema é determinada pela adequação que a combinação de elementos de interação obtêm frente às necessidades/capacidades da comunidade de usuários do sistema.

Além dos componentes da sintaxe abstrata, responsáveis pela descrição da árvore de sintaxe abstrata, a IU possue um conjunto de componentes sintáticos que descrevem a representação visual dos elementos de interação e a representação externa dos retornos semânticos (associados com a execução das tarefas). A descrição destas representações é chamada de sintaxe concreta.

A sintaxe concreta que descreve os elementos de interação é dita sintaxe concreta de entrada, pois descreve os elementos que irão permitir a geração dos tokens de entrada da IU. A sintaxe concreta que descreve as

representações externas dos retornos semânticos das tarefas é dita sintaxe concreta de saída, pois descreve como devem ser apresentados ao usuário os resultados das execuções das tarefas.

O acoplamento entre os dois conjuntos de componentes sintáticos (abstratos e concretos) é necessário para que possa ser determinado qual elemento de interação será utilizado em qual ponto de interação (na geração dos tokens de entrada) e qual representação externa será utilizada para a apresentação de qual retorno semântico (na apresentação dos tokens de saída).

Uma forma de análise de uma IU, é pela divisão de seus componentes "conversacional" e "processador" de tokens conforme o apresentado na figura 4.2.

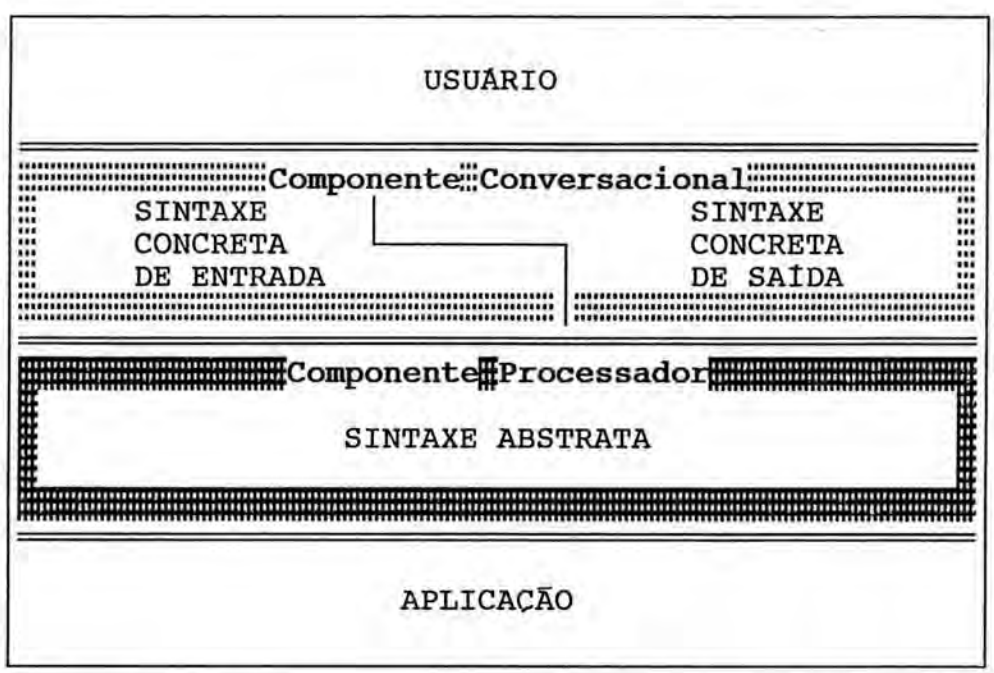

Figura 4.2 Componentes de uma visão lingüística da Interface

Nesta divisão, toda a comunicação com o usuário é feita pelo componente conversacional e toda a interpretação dos tokens e controle de execução da aplicação é feita pelo componente processador. A comunicação do usuário com o componente conversacional é realizada através de operações sobre os elementos de interação descritos pela sintaxe concreta de entrada. Estas operações vão gerar os tokens de entrada. Por outro lado, toda comunicação com o usuário é feita por meio das apresentações dos tokens de saída (retorno semântico das tarefas) e dos elementos de interação. A comunicação entre os componentes conversacional e processador é feita por meio dos tokens. Os tokens de entrada realizam a comunicação do componente conversacional com o processador. Os tokens de saída, gerados pelas funções da aplicação, permitem a comunicação do componente processador com o conversacional. A comunicação entre o componente processador e a aplicação é feita com chamadas às funções que executam as tarefas disponíveis, e a comunicação entre a aplicação e o componente processador é feita pelos retornos das chamadas das funções. Estes retornos determinam os tokens de saída que devem ser apresentados.

O componente "processador" é o responsável pela manipulação dos tokens da IU e pelo controle de execução da aplicação, ou seja, reconhece os tokens de entrada, os tokens de saída, as sentenças de entrada e controla a execução das tarefas da aplicação.

O componente "conversacional" descreve os aspectos relativos à definição dos tokens utilizados na IU, ou seja, descrevem as características físicas, como localização na tela, cores, padrões, molduras, representações gráficas, nomes de itens, etc utilizados

pelas apresentações dos estilos de interação e pelos retornos das tarefas executadas (tokens de saída).

A figura 4.3 apresenta a divisão da IU proposta por Coutaz [COU 85]. Como as IUs manipulam informações de entrada e de saída da aplicação, os elementos léxicos e sintáticos são sub-divididos em elementos de entrada e de saída, de acordo com as suas naturezas.

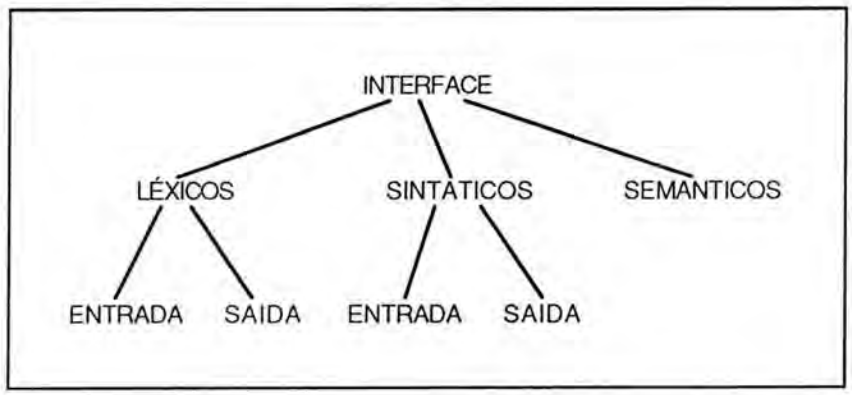

Figura 4.3 A estrutura da IU de Coutaz

Os componentes léxicos, sintáticos e semânticos das IUs tem, neste trabalho, as seguintes definições :

- **léxicos** : subdivididos em **tokens de entrada** e **de saída**, são formados por operações sobre os dispositivos de hardware. Os tokens de entrada são agrupados e apresentados ao usuário pelos elementos de interação. Os tokens de saída são associados às características dos dispositivos de saída do hardware por meio de regras [COU 85]. Isoladamente, os tokens não tem significado aparente. Os elementos léxicos de entrada são compostos por nomes de cores, itens, números, etc, e os elementos léxicos de saída são compostos por nomes de funções de saída.

- **sintáticos** : de uma forma geral os componentes sintáticos organizam os componentes léxicos (tokens) em **sentenças**. As interdependências lógicas definidas pelas sentenças não possuem significado em si, somente inter-relacionam os tokens. Os componentes sintáticos da IU são divididos em dois grupos. Um descreve os aspectos conversacionais da IU e o outro descreve os aspectos de processamento da IU. O grupo que descreve os aspectos conversacionais da IU é dividido, novamente, em dois conjuntos. Um descreve os elementos de interação utilizados pela IU e o outro descreve as representações dos tokens de saída. A descrição dos elementos de interação possui novamente componentes léxicos, sintáticos e semânticos. Os léxicos de descrição são compostos pelos valores que definem as posições ocupadas, cores utilizadas, janelas usadas, etc. Os sintáticos descrevem a seqüência em que devem ser feitas as descrições, e os semânticos são os retornos associados a cada um dos itens/comandos usados/manipulados pelo elemento. A descrição das representações dos tokens de saída também possue elementos léxicos, sintáticos e semânticos. Os léxicos de descrição das representações são formados

pelos nomes das funções que vão operar sobre os dispositivos de saída do hardware e pelos parâmetros utilizados. Os sintáticos descrevem as regras de composição para os tokens de saída (**sentenças de saída**) [COU 85] e os semânticos, novamente, são os significados que a execução destas representações tem para o usuário. Já o grupo que descreve os aspectos de processamento da IU define, numa gramática, todas as maneiras disponiveis aos usuários para alcançar (especificar) as tarefas da aplicação (**sentenças de entrada**), o fluxo de controle da execução da aplicação e todas as sentenças de saída geradas a partir dos retornos das execuções das tarefas.

- **semânticos** : Associam às sentenças de entrada as identificações das funções que deverão ser executadas pela aplicação para a execução da tarefa. A tarefa a ser executada é determinada pela sentença formada pelos tokens de entrada, gerados a partir das operações do usuário sobre os elementos de interação [COU 85]. Os tokens de saída são associadas ao retorno da execução das tarefas.

4.2 Informações para Sintaxe Concreta de Entrada

Para a descrição dos elementos de interação, fazem-se necessários a identificação dos componentes léxicos, sintáticos e semânticos de definição. De forma geral, os componentes léxicos são formados por valores que descrevem as caracteristicas físicas de apresentação dos elementos de interação. Os componentes sintáticos determinam o formato construtivo das declarações dos elementos de interação, especificando quais caracteristicas são obrigatórias e quais são opcionais. Por último, os componentes semânticos determinam quais são os retornos que

devem ser gerados quando é especificado, pelo usuário, algum comando descrito no elemento. Os retornos semânticos podem ser interpretados como tokens de entrada para a sintaxe abstrata (componente processador da IU), ou como invocação de outro elemento de interação. O significado que cada token de entrada gerado tem é determinado pela sintaxe abstrata da IU. Os tokens de entrada podem ser identificações de tarefas ou valores de parâmetros de tarefas.

4.2.1 Descrição de Menus

Na descrição de elementos de interação que implementam o estilo de menus, são necessárias a descrição das suas características físicas e dos retornos que devem gerar, assim como os nomes externos que os identificam ao usuário. A figura 4.4 lista os elementos léxicos necessários para a descrição de um menu. O formato textual da descrição é determinado pelo componente sintático da sintaxe concreta de entrada. A semântica é determinada pelos tipos dos retornos gerados quando da seleção de algum nome externo. É o gerenciador da interação quem identifica se o retorno gerado é outro elemento de interação ou um token de entrada.

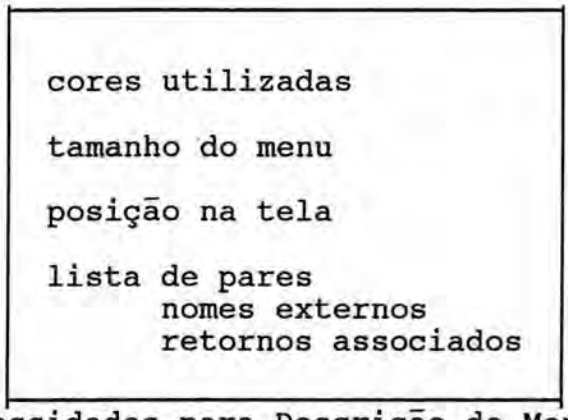

Figura 4.4 Necessidades para Descrição de Menus

4.2.2 Descrição de Formulários

Na descrição de elementos de interação que implementam o estilo de formulários, são necessárias a descrição das suas características físicas, dos campos de parâmetros a serem informados, dos rótulos de auxílio a identificação dos parâmetros e dos retornos que devem gerar, assim como os rótulos e a(s) ação(ões) de seleção que os identificam ao usuário. A figura 4.5 lista os elementos léxicos necessários para a descrição de um formulário. O formato textual da descrição é determinado pelo componente sintático da sintaxe concreta de entrada. A semântica é determinada pelos tipos dos retornos gerados quando da execução de alguma ação de seleção dos rótulos dos retornos. É o gerenciador da interação quem identifica se o retorno gerado é outro elemento de interação ou um token de entrada.

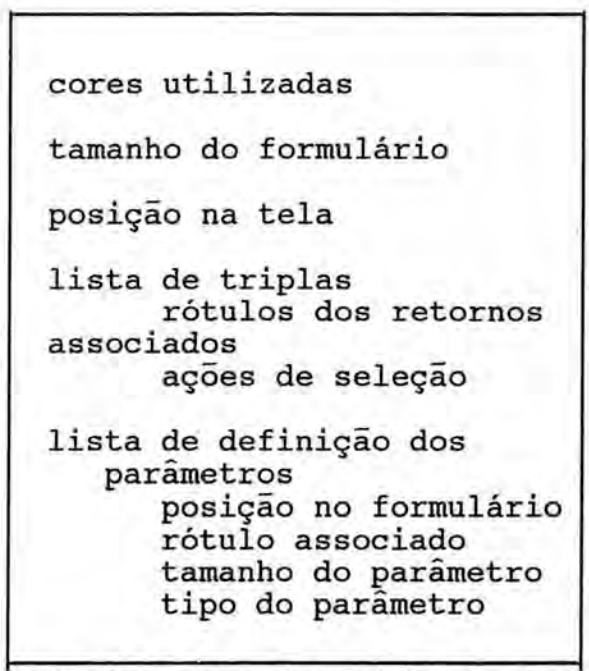

Figura 4.5 Necessidades para Descrição de Formulários

4.2.3 Descrição de Linguagem de Comandos

Na descrição de elementos de interação que implementam o estilo de linguagem de comandos, são necessárias a descrição das suas características físicas, da gramática que define a sintaxe dos comandos, do rótulo que indica a espera de uma entrada e dos retornos que deve gerar. A figura 4.6 lista os elementos léxicos necessários para a descrição de uma linguagem de comandos. O formato textual da descrição é determinado pelo componente sintático da sintaxe concreta de entrada. A semântica é determinada pelos tipos dos retornos gerados quando da especificação de alguma entrada digitada pelo usuário e reconhecida pela gramática que define a linguagem. É o gerenciador da interação quem identifica se o retorno gerado é outro elemento de interação ou um token de entrada.

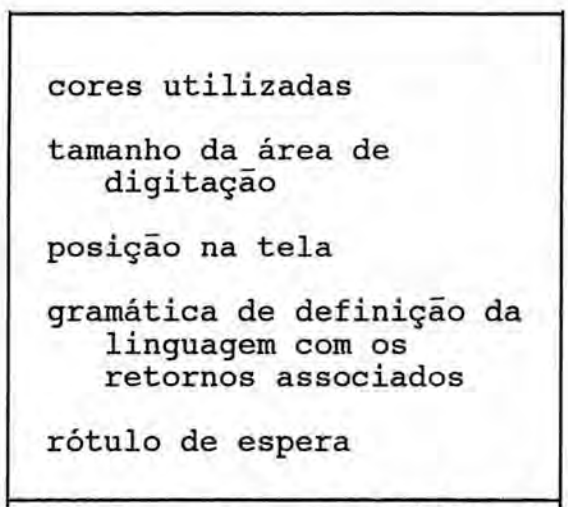

Figura 4.6 Necessidades para Descrição de Linguagem de Comandos

4.2.4 Descrição de Comandos Diretos

Na descrição de elementos de interação que implementam o estilo de comandos diretos, são necessárias a descrição das características físicas da figura associada,

da figura em si, dos retornos que devem gerar para cada operação executada sobre a figura a relação de operações válidas. A figura 4.7 lista os elementos léxicos necessários para a descrição de um menu. O formato textual da descrição é determinado pelo componente sintático da sintaxe concreta de entrada. A semântica é determinada pelos tipos dos retornos gerados quando da execução das operações sobre a figura que representa o elemento. É o gerenciador da interação quem identifica se o retorno gerado é outro elemento de interação ou um token de entrada.

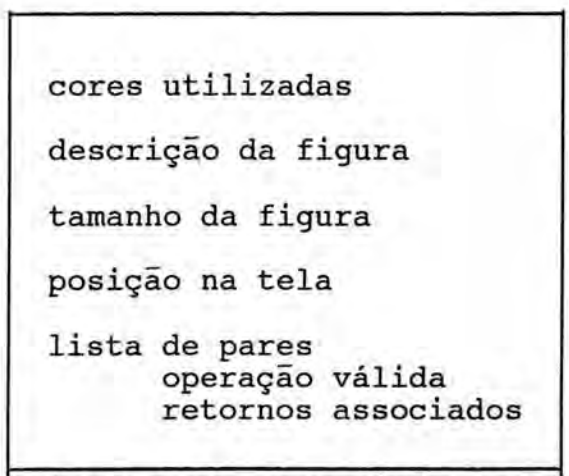

Figura 4.7 Necessidades para Descrição de Comando Direto

4.3 Informações para Sintaxe Concreta de Saída

A sintaxe concreta de saída é definida por um conjunto de regras que associam aos tokens de saída seqüências de chamadas a funções que realizam operações sobre os dispositivos de saída.

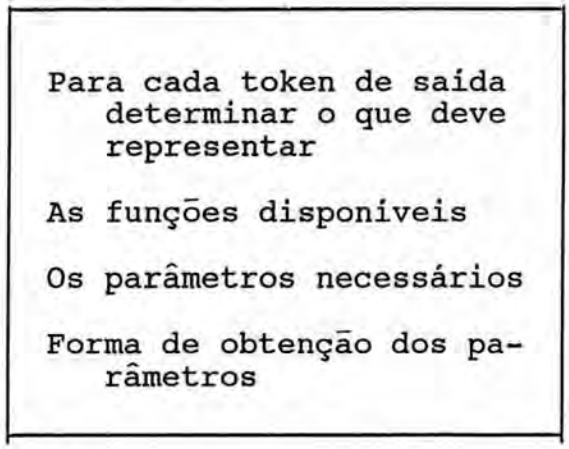

Figura 4.8 Necessidades da Sintaxe Concreta de Saída

A figura 4.8 lista as informações necessárias para a definição das regras de representação dos tokens de saída.

4.4 Informações para Sintaxe Abstrata

A sintaxe abstrata trabalha com os tokens de entrada gerados pela sintaxe concreta de entrada para determinar o fluxo de execução da aplicação e com os retornos semânticos gerados pela execução das funções da aplicação para identificar os tokens de saída que devem ser apresentados ao usuário. A figura 4.9 lista as necessidades de informação que a definição da sintaxe abstrata de uma IU requer.

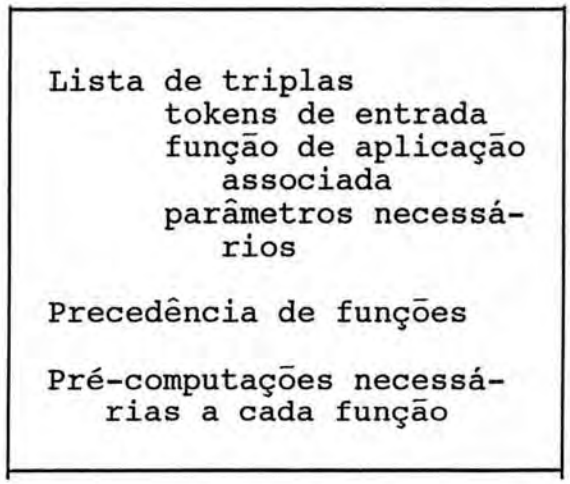

Figura 4.9 Necessidade da Sintaxe Abstrata

5 A PROPOSTA

Este capítulo apresenta uma extensão do formalismo de gramática de atributos (GA) através da agregação de seções que permitem a descrição dos componentes conversacional e processador da IU (ver 4.1). Na seção 5.1 são apresentados os conceitos que definem formalmente uma gramática de atributos. A seção 5.2 descreve as extensões necessárias à criação da linguagem proposta a partir de uma GA. Das seções 5.3 até a 5.8 são descritos, detalhadamente, os objetivos e usos da linguagem proposta.

5.1 Definição de uma Gramática de Atributos

Gramáticas de Atributos (GAs) são obtidas pela extensão do formalismo de Gramáticas Livres de Contexto (GLC) [HOR 86]. Por definição uma GLC é uma quádrupla (**N**,**T**,**P**,**S**) onde :

- **N** é um conjunto de símbolos não terminais
- **T** é um conjunto de símbolos terminais
- **P** é um conjunto de produções
- **S** é o símbolo inicial da gramática;

No escopo deste trabalho, especial atenção é destinada às GLCs de descrição de IUs. Para este fim, são estendidas com a incorporação de ações semânticas [GRE 86]. Ações semânticas são formadas por equações simples e chamadas a rotinas da aplicação. As principais características de uma GLC assim expandida são:

- os terminais da gramática são os tokens de entrada da IU, representando as ações do usuário;
- as produções da gramática definem a estrutura do diálogo empregado pelo usuário em sua interação com a aplicação;
- ações semânticas são associadas às produções da gramática e executadas para o atendimento das interações do usuário.

Por definição, uma GA é uma tripla (**G**,**A**,**E**) [SCH 90], onde:

- **G** é uma GLC;
- **A** é um conjunto de atributos onde

  A(X) = atributos para X pertencente a **N U T** e

  A(X) e A(Y) são disjuntos sse X <> Y;
- **E** é um conjunto de equações de atributos onde

  E(p) = equações para a produção p pertencente a P;

Uma característica das GAs é a possibilidade de troca de valores de atributos entre as produções. Os atributos que são "passados para baixo" na estrutura gramatical são ditos herdados pelas produções que vão ser reconhecidas, e as "passadas para cima" na estrutura são ditas sintetizadas pelas produções reconhecidas. É o

mecanismo de herança e sintetização de atributos que permite a verificação da semântica estática de programas[1].

Na prática, GAs tem sido usadas como um poderoso formalismo para a descrição da semântica de linguagens de programação. Técnicas incrementais de avaliação de atributos têm tornado possível a construção de ambientes de programação baseados em editores de linguagens [ESP 89]. O mecanismo de avaliação que permite a automática verificação e correção da semântica estática de programas em editores dirigidos pela sintaxe, sempre que alguma mudança é feita na sua árvore sintática abstrata [FAV 87], permite a automatização do controle do diálogo entre o usuário e a aplicação.

A similaridade entre os problemas relacionados ao reconhecimento incremental de programas e dos diálogos interativos, apontam para a possibilidade de uso de GAs para definição de IUs. Assim como no reconhecimento incremental de tokens, o reconhecimento de uma ação do usuário num diálogo interativo desencadeia uma série de ações da aplicação (e possíveis modificações da própria IU) que podem por sua vez desencadear outras, num processo de propagações sucessivas. Esta dependência entre ações do usuário e ações da aplicação (e da IU), direta e indiretamente desencadeadas, torna adequada a utilização de GA.

---

[1] Semântica estática é aquela que pode ser determinada em tempo de compilação das especificações, por exemplo o tipo das variáveis que participam de uma expressão. Semântica dinâmica é aquela que só pode ser determinada quando da execução da aplicação [AHO 86]. Um exemplo onde torna-se quase impossível a verificação da semântica estática são as "variáveis" do LISP, nesta linguagem o tipo das variáveis é indeterminado, podendo ser alterado ao longo da execução do programa.

A integração das rotinas da aplicação e a definição da IU é facilitada pelo uso de um mesmo formalismo. Scott Hudson mostrou, em seu sistema Planit, que apresentações podem ser computadas através de atributos [HUD 86]. No entanto, apesar de usar GA para apresentações, Hudson fez uso também de diagramas de transição de estados para manipular/descrever os diálogos. Isto implica na existência de um formalismo hibrido para descrição de IU : um para descrever o componente conversacional (a apresentação) e outro para descrever o componente processador (a estrutura do diálogo e as chamadas às rotinas da aplicação).

Quando o usuário realiza uma ação, o retorno associado é refletido pela mudança do que lhe é exibido. Para esta mudança, utiliza-se as informações de apresentação contidas nas seções que descrevem a apresentação da IU.

5.2 Extensões Aplicadas

Para que uma GA permita a definição do componente conversacional da IU de forma independente do componente processador das ações do usuário, é necessário que lhe sejam agregadas algumas extensões. Estas podem ser divididas em três grupos. As seções do primeiro grupo descrevem um conjunto de informações de apoio, usadas pelas seções do segundo e do terceiro grupo. As seções do segundo grupo descrevem as apresentações físicas, os tokens de entrada gerados pelos estilos de interação e os tokens de saída usados para traduzir os retornos associados às ações do usuário processadas nas seções do terceiro grupo. Finalmente, as seções do terceiro grupo descrevem a

estrutura do diálogo, as funções da aplicação associadas às ações do usuário e indicam os estilos de interação disponíveis ao usuário. A GA assim expandida, e aqui proposta, tem o nome de GRamática de Atributos Estendida para a Definição de Interfaces com o Usuário (GRAEDIUS). Uma primeira tentativa de definição encontra-se em [SCH 90].

As seções que permitem a definição das informações de apoio aos outros grupos e que compõe o primeiro grupo são :

- seção "CONJUNTOS" : nesta seção são declarados os conjuntos de caracteres que auxiliam na definição dos tokens de entrada;
- seção "AUTÔMATOS" : esta seção define os autômatos finitos que descrevem os tokens de entrada;
- seção "TOKENS" : nesta seção são declarados, explicitamente, os tokens de entrada utilizados pela seção "SINTAXE ABSTRATA & SEMÂNTICA" (ver a frente, no 3º grupo;
- seção "ASSINATURAS" : nesta seção são declarados os formatos de chamada de cada uma das funções estranhas ao ambiente proposto, incluindo-se as funções da aplicação;
- seção "APRESENTAÇÃO" : esta seção define os conjuntos de cores, as janelas e as máscaras de edição utilizadas pela seção de geração do segundo grupo. É sub-dividida em :
  - sub-seção "CORES" : define os conjuntos de cores válidas para cor da letra, cor de fundo, cor de

borda, cor da sombra, cor do caractere de identificação da opção (no caso de menus) e a cor que identifica a indisponibilidade da opção (no caso de menus e formulários);

- sub-seção "JANELAS" : esta sub-seção define as áreas da tela que vão estar disponíveis para uso pelas funções da aplicação, pelos estilos de interação e pelos tokens de saída;

- sub-seção "MÁSCARAS" : nesta seção são definidas as máscaras que vão permitir a edição dos valores informados nos formulários.

As seções de descrição das apresentações visuais dos estilos de interação e dos tokens de saída que compõe o segundo grupo são :

- seção "GERAÇÃO" : nesta seção são descritos os estilos de interação, mas como são três os estilos previstos pela proposta, é sub-dividida em :

  - sub-seção "MENUS" : descreve a apresentação visual (localização, cores e janela) usadas pelos menus, os nomes dos itens que serão apresentados ao usuário e ou os tokens de entrada gerados ou os nomes dos sub-menus invocados como retorno às ações do usuário;

  - sub-seção "FORMULÁRIOS" : esta sub-seção descreve a apresentação visual (localização, cores e janela) usadas pelos formulários, os nomes das tarefas associadas, os rótulos dos campos esperados/informados e a descrição de cada campo utilizado;

- sub-seção "LINGUAGEM DE COMANDOS" : nesta sub-seção são descritas a apresentação visual (localização, cores e janelas) onde o usuário digitará os comandos que deseja executar.

As seções que descrevem a estrutura do diálogo e as associações entre este e os estilos de interação definidos compõe o terceiro grupo e são :

- seção "REGRAS DE ASSOCIAÇÃO" : esta seção permite a determinação dos estilos de interação válidos para cada ponto de interação do diálogo;
- seção "SINTAXE ABSTRATA & SEMÂNTICA" : esta seção descreve a estrutura do diálogo adotada pelo sistema e, em suas ações semânticas, as funções da aplicação que devem ser executadas para cada ação do usuário.

A divisão das extensões agregadas nos três grupos apresentados, permite que o componente conversacional da IU seja definido independentemente do componente de processamento. O componente conversacional é definido pelas seções do 1º e 2º grupo, enquanto que o componente processador e a ligação entre eles são definidos no 3º grupo. A separação destes componentes permite vislumbrar a possibilidade de que uma aplicação tenha mais de um conjunto de estilos de interação associado. Tal característica pode ser facilmente incorporada devido à flexibilidade fornecida pela proposta GRAEDIUS. Deve-se ressaltar que a troca de estilos é factível de controle, enquanto que alterações que reflitam mudanças da estrutura do diálogo não são facilmente manipuladas. A dificuldade esta em estabelecer as correspondências entre as estruturas, de modo a garantir

suas equivalências. A alteração da estrutura do diálogo equivale a criar uma nova aplicação.

Alguns componentes da IU tem sua especificação dispersa por várias seções da GRAEDIUS (menus = cores + janelas + tokens), enquanto que algumas seções podem especificar, totalmente, alguns componentes da IU (estrutura de diálogo = sintaxe abstrata & semântica). Além disto, o controle do diálogo pode ser gerenciado dinamicamente através da simples avaliação dos atributos.

Poder mesclar especificações, tem como grande vantagem o fato de permitir um desenvolvimento quase que incremental da aplicação. O ciclo de desenvolvimento normalmente seguido numa aplicação não considera muito os aspectos da IU [BET 87], enquanto que o induzido por GRAEDIUS reverte este quadro, ou seja, leva o desenvolvimento inicial da IU, através de um protótipo, e depois esta é "decorada" com os procedimentos/funções da aplicação.

O ambiente GRAEDIUS prevê a existência de uma biblioteca de funções. As funções são bastante simples, executando desenho de retas, arcos de círculo, círculo, escrita na tela, etc. O analista/programador da IU pode utilizá-las ou fazer uso de rotinas mais complexas desenvolvidas por outrem. A vantagem da utilização das rotinas fornecidas é a dispensa de declaração dos seus domínios, tanto de entrada (parâmetros) quanto os de saída (retornos).

5.3 Seções do Primeiro Grupo

5.3.1 Conjuntos

Nesta seção são descritos os conjuntos de códigos que auxiliam na definição dos tokens de entrada utilizados pela GRAEDIUS. Estes conjuntos formam um artifício simplificador do processo de definição dos autômatos. Cada conjunto é composto por definições de caracteres ou códigos ASCII agrupados. O agrupamento destes códigos pode ser feito de três formas distintas :

- cada caractere individual é delimitado por aspas duplas ('"');
- cada código ASCII é representado por seu número no padrão ASCII precedido do símbolo "cerca" ('#');
- intervalos são representados pela presença de dois pontos seguidos ('..') entre os extremos do intervalo ("a"..#250).

A sintaxe que descreve o formato da definição de conjuntos é dada pela gramática que segue :

```
LEX_CONJUNTO    -> "conjuntos" LST_CONJUNTO
                |  .
LST_CONJUNTO    -> IDENT "->" CORPO_CJTO ";" LST_CONJUNTO
                |  .
CORPO_CJTO      -> CADEIA OUTRA_CADEIA CORPO_CJTO_
                |  CARACTERE OUTRO_CAR CORPO_CJTO_
                |  IDENT CORPO_CJTO_ .
CORPO_CJTO_     -> "," CORPO_CJTO
                |  .
OUTRA_CADEIA    -> ".." CADEIA
                |  .
```

```
OUTRO_CAR       -> ".." CARACTERE
                |  .
```

Um exemplo de definição de conjuntos é :

```
Conjuntos

  Digitos     -> 0..9;
  LetrasMin   -> "a".."z";
  LetrasMai   -> "A".."Z";
  Letras      -> LetrasMin,LetrasMai;
  Qualquer    -> Letras,Digitos;
```

Neste exemplo os nomes **Digitos**, **LetrasMin**, **LetrasMai**, **Letras** e **Qualquer** são nomes de conjuntos, enquanto que os lados direitos das flechas ("->") são as definições associadas.

5.3.2 Autômatos

A seção de autômatos descreve os autômatos finitos que descrevem os tokens de entrada. Neste mapeamento são usadas produções de gramática regular, onde as "cabeças" das produções representam os estados dos autômatos e os "corpos" das produções fazem as vezes dos vários caminhos que levam de um "estado" a outro. Cada opção de caminho é representada por uma distinta possibilidade de "corpo" da produção. A sintaxe usada para a descrição dos autômatos é dada pela gramática a seguir :

```
LEX_AUTOMATO    -> "automatos" RGR_AUTOMATO
                |  .
RGR_AUTOMATO    -> IDENT "->" CORPO_AUTOMAT ";" RGR_AUTOMATO
                |  .
CORPO_AUTOMAT   -> PRIM_ELEM_AUT SEG_ELEM_AUT CRPO_AUTOMAT
                |  .
CRPO_AUTOMAT    -> "|" CORPO_AUTOMAT
                |  .
```

```
PRIM_ELEM_AUT  -> IDENT
                | CARACTERE
                | CADEIA .
SEG_ELEM_AUT   -> IDENT
                | .
```

A principal característica desta sintaxe é o fato de que cada produção tem no máximo dois elementos em cada um de seus corpos. Os valores válidos para a produção <PRIM_ELEM_AUT> são os conjuntos definidos na seção de conjuntos e/ou os caracteres/códigos explicitamente referenciados. Para a produção <SEG_ELEM_AUT> somente são válidos identificadores de outras produções.

Um exemplo de definição da seção de autômatos é :

```
Autômatos

  IDENT       -> Letras Corpo_Nome;
  Corpo_Nome  -> Qualquer Corpo_Nome | Qualquer;

  NOME        -> Ident;

  INTEIRO     -> Digito INTEIRO      | Digito;
```

Neste exemplo os nomes **Ident**, **Nome**, e **Inteiro** são os nomes dos estados iniciais dos autômatos de reconhecimento e **Corpo_Nome** é um estado intermediário.

5.3.3 Tokens

A declaração explícita dos tokens de entrada é necessária para que o reconhecedor de GRAEDIUS identifique, na estrutura de diálogo definida para a IU, os vários comandos e parâmetros que o componente processador tem condições de processar. Os tokens tem por função básica identificar as várias tarefas que estarão a disposição do

usuário, além dos tipos válidos de parâmetros. Os tokens constantes são os que identificam tarefas e são tratados como "palavras reservadas" pelo executor de GRAEDIUS, enquanto os parâmetros são tratados como tokens variáveis de uma linguagem, tendo sua estrutura definida na seção de autômatos (ver 5.3.2). A sintaxe utilizada na descrição dos tokens é dada pela gramática a seguir :

```
LEX_TOKEN       -> "tokens" TOKENS_LEX .
TOKENS_LEX      -> IDENT LST_PAL_RESERV ";" TOKENS_LEX
                |  .
LST_PAL_RESERV  -> "exceto" CADEIA PROX_PAL_RES
                |  .
PROX_PAL_RES    -> "," CADEIA PROX_PAL_RES
                |  .
```

Os autômatos descritos no exemplo da seção 5.3.2 teriam os tokens associados descritos conforme o exemplo que segue :

```
Tokens

  IDENTIFICADOR
       exceto "tarefa_aplic_1", "tarefa_aplic_2";
  NOME;
  INTEIRO;
```

Neste exemplo os nomes **Identificador**, **Nome**, e **Inteiro** são os nomes dos tokens utilizados na seção de sintaxe abstrata&semântica (ver 5.8). Estes são criados pelos estilos de interação definidos na seção de geração (ver 5.6), enquanto as exceções (**tarefa_aplic_1** e **tarefa_aplic_2**) são os tokens constantes que identificam as tarefas que o usuário tem a disposição.

5.4 Seção de Assinaturas

A seção de assinaturas tem como objetivo permitir a definição dos domínios das funções estranhas à biblioteca de funções da GRAEDIUS. Se a IU utilizar alguma função externa, torna-se necessário a declaração dos seus domínios usando a sintaxe apresentada na gramática abaixo :

```
ASSINATURAS     -> "assinaturas" CORPO_ASSINAT FIM_SECCAO
                |  .
CORPO_ASSINAT   -> DECL_FUNC CORPO_ASSINAT
                |  .
DECL_FUNC       -> ESPEC_DECLAR IDENT DECL_PARM ";" .
DECL_PARM       -> "(" LST_PARM ")"
                |  .
LST_PARM        -> PARAMETRO LST_PARM_
                |  .
LST_PARM_       -> "," PARAMETRO
                |  .
PARAMETRO       -> CL_CONST CL_MOD_SINAL CL_MOD_TAM CL_TIPO
                   POINTER NOME_PARM LST_PARM_
                |  "..." .
CL_CONST        -> "const"
                |  .
CL_MOD_SINAL    -> "signed"  |  "unsigned"  |  .
CL_MOD_TAM      -> "short"   |  "long"      |  .
CL_TIPO         -> "char"    |  "int"       |  "void"
                |  "float"   |  "double"    |  IDENT .
POINTER         -> CL_MOD_POINTER "*" POINTER_
                |  .
POINTER_        -> "*" POINTER_
                |  .
CL_MOD_POINTER  -> "far"     |  "near"      |  "huge"    |  .
NOME_PARM       -> IDENT
                |  .
```

A declaração dos domínios da função **f1** que recebe como parâmetros um inteiro, um número de ponto flutuante e retorna um inteiro tem a seguinte declaração na seção de assinaturas :

Assinaturas

```
int  f1( int, float);
void linha( int; int);
```

O formato de declaração das assinaturas é bastante idêntico ao utilizado pela linguagem C. Optou-se por este tipo de declaração por não ser nada estranho aos programadores. Se fosse novo, geraria a necessidade de treinamento extra.

5.5 Seção de apresentação

As informações definidas na seção de apresentação auxiliam a definição dos estilos de interação e o(s) local(is) onde será(ão) apresentado(s) os tokens de saída. É subdividida nas sub-seções de definição de cores, janelas e máscaras.

5.5.1 Cores

Permite a associação de identificadores a nomes de cores que devem ser atribuídas às cores do fundo, da borda e da sombra da janela, das letras normais e do caractere de identificação e de desabilitação da opção (no caso de menu) ou campo não habilitado (no caso de formulário). A estrutura notacional adotada é a mesma utilizada pelo GIM [BAG 89], um gerador de interfaces baseado em menus desenvolvido por alunos da graduação do curso de ciências da computação da UFGRS.

A gramática que descreve a sintaxe de definição dos conjuntos de cores válidos e disponíveis é :

```
APRE_CORES      -> "cores" LST_CORES
                |  .
LST_CORES       -> IDENT "=" /* nome do conj. de cores */
                   IDENT "," /* cor de letra          */
                   IDENT "," /* cor de fundo         */
                   IDENT "," /* cor de borda         */
                   IDENT "," /* cor de sombra        */
                   IDENT "," /* cor de car-id        */
                   IDENT ";" /* cor de off           */
                   LST_CORES_ .
LST_CORES_      -> LST_CORES
                |  .
```

A definição de três distintos conjuntos de cores é o exemplo apresentado a seguir :

```
Cores
    /*      letra,  fundo,   borda,sombra,car-id,  off /
  Cor1 =     azul,amarelo,vermelho, verde,branco, cinza;
  Cor2 = vermelho,   azul,   verde, cinza,  azul, preto;
  CorH =     azul,amarelo,    azul, preto, preto,  azul;
```

5.5.2 Janelas

Define as áreas da tela passiveis de utilização pelos estilos de interação e pelas apresentações dos tokens de saída. A sintaxe que define o formato de definição das janelas é dada pela gramática que segue :

```
APRE_JANELAS    -> "janela" IDENTIFICACAO TAM_JAN ID_CORES
                   POS_JAN APRE_JANELAS_ .
APRE_JANELAS_   -> APRE_JANELAS
                |  .
IDENTIFICACAO   -> IDENT ";" .
TAM_JAN         -> "area" "=" INTEIRO "," INTEIRO ";" .
ID_CORES        -> "cor"  "=" IDENT ";"
                |  .
POS_JAN         -> "posicao" "=" INTEIRO "," INTEIRO ";"
                |  .
```

Um exemplo de declaração de janelas é :

```
Janela JMenu0;     /* tem o nome de "JMenu1"          */
  area  = 15,4;    /* 15 colunas por 4 linhas         */
  cor   = Cor1;    /* usa o conjunto de cores "Cor1" */
```

Neste exemplo o nome **JMenu0** identifica esta janela, com uma área útil de 15 colunas de largura e 4 linhas de altura, tendo válido como "default" o conjunto de cores **cor1**.

5.5.3 Máscaras

Nesta subseção são definidas as máscaras utilizadas nos formulários, para o recebimento dos campos. As definições das máscaras não prevêem a especificação de técnicas de abreviaturas, sinônimos, etc. Além de permitir a determinação dos locais ocupados por caracteres especiais obrigatórios, o único mecanismo suportado é o que permite a combinação de várias máscaras na criação de outras. Esta mecanismo de concatenação é representada pelo simbolo "&". A gramática que descreve a sintaxe de definição das máscaras é :

```
MASCARAS        -> "mascaras" LST_MASCARAS
                |  .
LST_MASCARAS    -> IDENT "=" MASCARA ";" LST_MASCARAS
                |  .
MASCARA         -> CADEIA MASCARA_
                |  IDENT MASCARA_ .
MASCARA_        -> "&" MASCARA
                |  .
```

Na definição de máscaras, cada caractere sublinhado ("_") corresponde a um caractere a ser digitado pelo usuário e todo caractere diferente de sublinhado é interpretado como uma constante. As máscaras para a definição de datas apresentadas no exemplo que segue, são um bom exemplo de sua definição.

Mascaras

```
dia    = "__";
mes    = "__";
ano    = "__";
data_e = dia & "/" & mes & "/" & ano;
data_i = mes & "-" & dia & "-" & ano;
```

5.6 Seção de Geração

A seção de geração é subdividida em dois grupos. No primeiro grupo, subseções definem os estilos de interação que serão utilizados pela IU e o segundo grupo descreve as apresentações visuais dos tokens de saída gerados como retorno às ações do usuário. As subseções do primeiro grupo permitem a definição dos estilos de interação de menus, formulários e linguagens de comando. Todos estes estilos implementam a metáfora conversacional (ver 3.1.1).

Uma característica comum a todos os elementos definidos na seção de geração é a capacidade de utilizarem no corpo de suas descrições referências à atributos. A grande vantagem advinda do uso deste artifício é a flexibilidade obtida na distribuição dos elementos na tela. Esta flexibilidade permite que o reconhecedor de GRAEDIUS tenha maior controle sobre a IU, além de possibilitar que, com algum exercício, sejam simuladas técnicas assíncronas de interação.

5.6.1 Menus

Nesta subseção são definidos os menus utilizados pela aplicação. Cada menu definido possue um nome único de

identificação e informações de aspectos como sua posição, a janela associada, a direção, o conjunto de cores válido. A descrição dos itens apresentados possui informações como : o nome externo, ordem de apresentação, o retorno em caso de seleção e, opcionalmente, o rótulo que identifica o texto de ajuda dentro do arquivo de ajuda associado à aplicação. O retorno pode ser tratado de duas diferentes formas pelas funções de GRAEDIUS. Se identificar um nome de elemento de geração (menu, formulário ou linguagem de comando), indica que este deverá ser invocado (ativado), caso contrário o retorno é tratado como um token de entrada e inserido na memória do reconhecedor léxico. Em menus, os valores possíveis dos parâmetros de uma opção, se apresentados como menu, devem ser declarados explicitamente como itens deste submenu. A sintaxe de definição da subseção de menus tem a gramática apresentada a seguir :

```
MENU            -> "menu" IDENT_GER
                           AREA CORES DIR_MENU POSIC_MENU
                           OPCAO_DEFAULT NOME_AJUDA
                           OPCOES_MENU .
IDENT_GER       -> IDENT DECL_PARM ";" .
DECL_PARM       -> "(" LST_PARM ")"
                |  .
LST_PARM        -> PARAMETRO LST_PARM_
                |  .
LST_PARM_       -> "," PARAMETRO
                |  .
PARAMETRO       -> CL_CONST CL_MOD_SINAL CL_MOD_TAM
                   CL_TIPO POINTER NOME_PARM LST_PARM_
                |  "..." .
CL_CONST        -> "const"   |  .
CL_MOD_SINAL    -> "signed"  |  "unsigned"  |  .
CL_MOD_TAM      -> "short"   |  "long"      |  .
CL_TIPO         -> "char"    |  "int"       |  "void"
                |  "float"   |  "double"    |  IDENT .
POINTER         -> CL_MOD_POINTER "*" POINTER_
                |  .
POINTER_        -> "*" POINTER_
                |  .
CL_MOD_POINTER  -> "far"   |  "near"
                |  "huge"  |  .
NOME_PARM       -> IDENT
```

```
                |  .
AREA            -> "local"    "=" IDENT ";" .
CORES           -> "cor"      "=" VAR_OU_CADEIA ";" .
DIR_MENU        -> "direcao" "=" DIRECAO ";" .
DIRECAO         -> "h" |  "horizontal"
                |  "v" |  "vertical"
                |  IDENT .
POSIC_MENU      -> POSICAO
                |  .
POSICAO         -> "posicao" "=" VAR_OU_INT "," VAR_OU_INT
                       ";" .
VAR_OU_INT      -> IDENT
                |  INTEIRO .
OPCAO_DEFAULT   -> "ativa" "=" VAR_OU_INT ";"
                |  .
NOME_AJUDA      -> "ajuda"    "=" CADEIA ";"
                |  .
OPCOES_MENU     -> "opcoes" OPCAO_MENU OPCAO .
OPCAO           -> OPCAO_MENU OPCAO
                |  .
OPCAO_MENU      -> VAR_OU_CADEIA
                   ","
                   NUM_CAR_IDENT /* No car. de Ident.     */
                   ","
                   POSIC_NO_MENU /* No ordem na visualiz. */
                   ","
                   ATIV_DESATIV  /* Indic. de ativo/desat.*/
                   ":" RET_MENU NOME_AJUDA .
NUM_CAR_IDENT   -> VAR_OU_INT .
POSIC_NO_MENU   -> VAR_OU_INT .
ATIV_DESATIV    -> IDENT
                |  "f" | "falso"
                |  "v" | "verdadeiro" .
VAR_OU_CADEIA   -> IDENT
                |  CADEIA .
RET_MENU        -> IDENT_RET
                |  CADEIA ";" .
IDENT_RET       -> IDENT PARM_OPCIONAL ";" .
```

Um exemplo de definição de menus horizontais e verticais é :

```
menu Menu0 ;
  local    = JMenu0;
  cor      = DEFAULT;
  direcao = "h";
  posicao = 1,1;
  ativa    = 2;
```

```
  ajuda   = "menu0    ";  /* ind. texto de ajuda global */
                          /* ao menu */
  opcoes
      "Gerenciamento", 1, 1, v : menu1 (TRUE);
                          ajuda = "tarefas p/ gerenciar ";
      "Utilitarios"  , 1, 2, v : menu2 ;
                          ajuda = "uso dos utilitarios  ";
      "Relatorios"   , 1, 3, v : menu3 ;
                          ajuda = "como pedir relatorios";
      "Fim"          , 1, 4, v : menu4 ;
                          ajuda = "fim                  ";

menu Menu1 (bool hab) ;
  local   = JMenu1 ;
  direcao = vertical;
  opcoes
       "Inclusao" , 1, 1, hab : inclui_ficha ;
       "Alteracao", 1, 2, hab : altera_ficha ;
       "Exclusao" , 1, 3, hab : exclui_ficha ;
```

Nos exemplos apresentados, os nomes de identificação são : a)**Menu0** e b)**Menu1**.

O Menu0 está definido para aparecer na janela JMenu0, utilizar o conjunto de cores DEFAULT, ser apresentado como um menu horizontal ("h") (forma "default" de orientação), na posição absoluta de tela 1,1 (coluna = 1 e linha = 1), tendo a segunda opção como a ativa quando da primeira apresentação do menu. A ajuda global, indicada pela cláusula "ajuda", está associada ao rótulo "menu0" do arquivo de ajuda padrão, apresentando os itens : a) "Gerenciamento", como primeiro item do menu. Se selecionado acionará a execução do Menu1 e terá associada a ajuda identificada pelo rótulo "tarefas p/ gerenciar"; b) "Utilitarios", será o segundo item do menu. Se selecionado acionará a execução do menu2 e terá associado a ajuda identificada pelo rótulo "uso dos utilitarios". Apresenta ainda os itens c) "Relatorios" e d) "Fim" que seguem o mesmo mecanismo de definição.

O Menu1 está definido para aparecer na janela JMenu1, e ser apresentado como um menu vertical ("vertical"), com os itens "Inclusao", "Alteracao" e "Exclusao" sendo mostradas nesta ordem ao usuário. Quando um destes itens for selecionado, será retornado, respectivamente, o token Inclui_Ficha, Altera_Ficha ou Exclui_Ficha. A posição de JMenu1 é calculada em relação à posição ocupada pelo item "pai" ("Gerenciamento") do menu menu0. Por "default" o item ativo será o primeiro.

O Menu1 possui um parâmetro que irá determinar a habilitação dos seus itens. Este parâmetro informa à rotina que controla a execução, apresentação e interação dos menus com o usuário, que a habilitação dos três itens deste menu será determinada em tempo de execução. Para um exemplo mais completo e detalhado veja o 6.

5.6.2 Formulários

Nesta subseção são definidos os formulários utilizados pela IU do sistema para a recepção dos parâmetros necessários às funções da aplicação. O tratamento dos retornos associados é idêntico ao dedicado aos retornos dos menus. As informações mínimas necessárias para a descrição de um formulário são : seu nome de identificação, área de exibição, a posição em que aparecerá na tela, os comandos válidos e a descrição dos campos de recepção das informações. Opcionalmente podem ser informados os parâmetros relacionados com o formulário, as cores válidas e o rótulo do texto de ajuda associado. Cada comando válido para o formulário deve informar a posição onde será escrito o texto que identificará o comando, o nome da seleção rápida do comando (por exemplo, o caractere correspondente a uma tecla) e o retorno associado. A

descrição de cada um dos campos de entrada deve conter informações de sua posição na janela de exibição, seu tipo (numérico inteiro ou ponto flutuante, alfabético, data ou booleano), seu tamanho (no caso tipo numérico ou alfabético) a máscara de edição utilizada e o valor "default". O uso de rótulos pode auxiliar o usuário a informar corretamente os campos necessários. Para a descrição dos rótulos de auxilio, é necessária a informação da sua posição na janela de apresentação e o texto associado. A sintaxe que define o formato de declaração dos formulários é :

```
FORM_FILL[1]      -> "formulario" IDENT_GER
                           AREA CORES POSICAO NOME_AJUDA
                           TAREFAS ENTRADAS .
TAREFAS           -> "comandos" TRFA_FF LST_TAREFAS .
LST_TAREFAS       -> TRFA_FF LST_TAREFAS
                   |  .
TRFA_FF           -> POSIC_LABEL CADEIA "," IDENT ":" RET_FF
                        ";" .
                         /* POSX      ,  POSY     */
POSIC_LABEL       -> "(" VAR_OU_INT "," VAR_OU_INT ")" .
RET_FF            -> IDENT_RET
                   |  CADEIA ";" .
ENTRADAS          -> "entrada" CARAC_ENTRADA LST_ENTRADA .
LST_ENTRADA       -> CARAC_ENTRADA LST_ENTRADA
                   |  .
CARAC_ENTRADA     -> POSIC_LABEL LABEL_NOME_CMP ";"
                        NOME_AJUDA .
LABEL_NOME_CMP    -> TIPO_CAMPO
                   |  CADEIA
                   |  IDENT .
TIPO_CAMPO        -> "char"  DESC_CAMPO DEFAULT_CHAR
                   |  "int"   DESC_CAMPO DEFAULT_INT
                   |  "float" DESC_CAMPO DEFAULT_FLOAT
                   |  "data"  DEFAULT_CHAR
                   |  "bool"  DEFAULT_BOOL .
DESC_CAMPO        -> TAMANHO_CAMPO MASCARA_CAMPO .
TAMANHO_CAMPO     -> "(" VAR_OU_INT PRECISAO ")" .
PRECISAO          -> "," VAR_OU_INT
MASCARA_CAMPO     -> "," IDENT
                   |  .
DEFAULT_CHAR      -> "," DEF_CHAR
```

[1] Os não-terminais utilizados na definição sintática dos formulários e que não aparecem listados, tem sua sintaxe definida de forma idêntica à utilizada na definição de menus.

```
                |  .
DEF_CHAR        -> CADEIA
                |  IDENT .
DEFAULT_INT     -> "," DEF_INT
                |  .
DEF_INT         -> INTEIRO
                |  IDENT .
DEFAULT_FLOAT   -> "," DEF_FLOAT
                |  .
DEF_FLOAT       -> IDENT
                |  FLOAT .
DEFAULT_BOOL    -> "," DEF_BOOL
                |  .
DEF_BOOL        -> "v" | "verdadeiro"
                |  "f" | "falso"
                |  IDENT .
```

Se uma função da aplicação tivesse a necessidade de saber o nome, endereço, salário, data de admissão, número de dependentes e estado civil, então a declaração de um formulário que recebesse como atributos herdados a linha e coluna onde deveria ser apresentado e que permitisse a entrada destas informações poderia ser :

```
formulario Ficha_Cadastral (int coluna; int linha);
  local   = JFormulario1;
  posicao = coluna,linha;
  comandos
       /* x=10 e y=5 */
       (10,5)  "confirma - <F10>", F10 : conf_ficha ;
                                   ajuda = "confirma   ";
       (25,5)  "sai - <ESC>"     , ESC : cancelar ;
                                   ajuda = "cancela    ";
  entrada
       (03,2) "nome     : ";
       (14,2) char  (50), masc_0;  ajuda = "form_1.nome";
       (03,3) "endereco : ";
       (14,3) char  (50), masc_0;
       (03,5) "salario  : ";
       (14,5) float (7,2),masc_3; /* 7 posic. na parte int.
                                     e 2 na parte frac.  */
       (03,7) "data admissao : ";
       (19,7) data;
       (03,8) "dependentes   : ";
       (19,8) int   (2),  masc_2;
       (40,8) "casado  : ";
       (50,8) bool;                ajuda = "form_1.casado";
```

No exemplo apresentado, o formulário tem o nome Ficha_Cadastral, local de exibição a janela JFormulario1 que aparecerá na posição determinada pelos parâmetros coluna e linha, informados durante a execução do sistema de aplicação. Os comandos que estarão habilitados neste formulário são :

1) de confirmação das informações já preenchidas (comando Conf_Ficha), apresentado na coluna 10 da linha 5 do formulário (última linha) e está associado com a tecla F10;

2) de cancelamento do uso do formulário (comando Cancelar), apresentado na coluna 25 da linha 5 do formulário (última linha) e está associado com a tecla ESC.

Os campos e rótulos de auxílio associados ao formulário apresentado são :

1) um campo com formato alfanumérico de 50 posições e máscara de digitação masc_0 para a informação do nome do funcionário. Pela posição visual ocupada no formulário, faz par com o rótulo "nome";

2) um campo com formato alfanumérico de 50 posições e máscara de digitação masc_0. Pela posição visual ocupada no formulário, faz par com o rótulo "endereço";

3) um campo com formato numérico de ponto flutuante de sete posições inteiras, duas casas decimais e máscara de digitação masc_3. Pela posição visual ocupada no formulário, faz par com o rótulo "salário";

4) um campo com formato data sem valor *default*. Pela posição visual ocupada no formulário, faz par com o rótulo "data admissão";

5) um campo com formato numérico inteiro com dois dígitos e máscara de digitação masc_2. Pela posição visual ocupada no formulário, faz par com o rótulo "dependentes"; e

6) um campo com formato booleano (aceita somente verdadeiro/falso ou sim/não, etc.). Os campos booleanos não necessitam de máscara de edição. Pela posição visual ocupada no formulário, faz par com o rótulo "casado";

Os campos nome e casado possuem texto de ajuda associados aos rótulos form1.nome e form1.casado, respectivamente.

5.6.3 Linguagem de Comando

A subseção de linguagem de comando, de forma diferente das anteriores, não determina os nomes dos comandos, nem os tipos de parâmetros necessários para a especificação de uma tarefa. Esta subseção declara obrigatoriamente a janela onde ocorrerá a entrada dos comando e opcionalmente o conjunto de cores válidas, o texto e o local do *prompt* associado e o rótulo que identifica a texto de ajuda. Os comandos e os parâmetros

são definidos e declarados junto com a estrutura do diálogo na seção de sintaxe abstrata&semântica (5.8).

A rotina que implementa deste estilo não necessita fazer nenhuma espécie de critica às entradas fornecidas, pois estas são feitas pelo reconhecedor da GRAEDIUS. O tempo de ativação deste estilo é determinado pelo reconhecimento total da produção com a qual esteja associado[1]. Se houver a necessidade de recursão na estrutura do diálogo, deve-se criar um nodo *dummy* na estrutura, que servirá somente para controlar a ativação/desativação do estilo. A sintaxe para a descrição deste estilo é dada pela gramática que segue.

```
LING_COM        -> "ling_comando" IDENT_GER
                          AREA POSICAO PROMPT
                          POS_DIGIT ";" .
AREA            -> "local"   "=" IDENT ";" .
POSICAO         -> "posicao" "=" VAR_OU_INT "," VAR_OU_INT
                        ";" .
PROMPT          -> "prompt" POSIC_LABEL NOME_PROMPT ";"
                |  .
POSIC_LABEL     -> "(" INTEIRO "," INTEIRO ")" .
NOME_PROMPT     -> VAR_OU_CADEIA
                |  .
                                        /* POSX , POSY */
POS_DIGIT       -> "digitar" "em" VAR_OU_INT "," VAR_OU_INT
                      ";"
                |  .
```

A definição de uma linguagem de comando pode ser feita como no exemplo que segue.

```
ling_comando Linguagem_01 (char texto);
  local   = JLComando01;
  posicao = 5,3;
  prompt  (1,1) *texto;
  digitar em 9,1;
```

---

[1] Os mecanismos de associação são explicados na seção 5.7.

5.6.4 Saída

Esta subseção, do segundo grupo da seção de geração, permite que as representações dos tokens de saída sejam alteradas com grande facilidade e sem necessidade de conhecimento da aplicação. As representações podem ser formadas por desenhos (linhas, pontos, circulos, etc), por uma composição destas formas ou por cadeias de caracteres. Esta capacidade confere grande flexibilidade e independência entre a aplicação e a forma de apresentação dos retornos associados com as ações do usuário.

Para descrever a forma de apresentação dos tokens de saída ao usuário, utiliza-se um conjunto de chamadas a funções de manipulação dos dispositivos de saída do hardware e um conjunto enumerado das janelas onde poderá ser exibido. A descrição da apresentação é formada por uma concatenação, não vazia, de chamadas a funções da biblioteca de GRAEDIUS ou da aplicação. O reconhecedor de GRAEDIUS trata estas chamadas de forma análoga, não fazendo distinção. A enumeração das janelas onde serão executadas as apresentações dos tokens é útil, pois permite que, através de somente uma regra de associação (ver 5.7), uma mesma representação de token de saída seja visualizada em mais de uma janela. Esta capacidade permite que num editor gráfico, o usuário visualize os objetos tanto no universo e na janela de trabalho [FOL 84]. A gramática que define a sintaxe de descrição das apresentações dos tokens de saída é :

```
GER_SAIDA       -> "saida" CORPO_SAIDA
                |  .
CORPO_SAIDA     -> SEQ_SAIDA CORPO_SAIDA_ .
CORPO_SAIDA_    -> CORPO_SAIDA
                |  .
```

```
SEQ_SAIDA       -> LOCAIS IDENT DECL_PARM "->" LST_FUNCOES
                       ";" .
LOCAIS          -> "[" LST_LOCAIS "]"
                |  .
LST_LOCAIS      -> IDENT LST_LOCAIS_ .
LST_LOCAIS_     -> "," LST_LOCAIS
                |  .
LST_FUNCOES     -> CALL_FUNC LST_FUNCOES_ .
LST_FUNCOES_    -> LST_FUNCOES
                |  .
CALL_FUNC       -> IDENT PARM_CHAMADA .
PARM_CHAMADA    -> "(" LST_PARM_CALL ")" .
LST_PARM_CALL   -> EXPR_CHAMADA LST_PARM_CALL_
                |  .
LST_PARM_CALL_  -> "," LST_PARM_CALL
                |  .
EXPR_CHAMADA    -> TERMO_EXPR EXPR_CHAMADA_ .
EXPR_CHAMADA_   -> ADDOP_EXPR TERMO_EXPR EXPR_CHAMADA_
                |  .
TERMO_EXPR      -> FATOR_EXPR TERMO_EXPR_ .
TERMO_EXPR_     -> MULOP_EXPR FATOR_EXPR TERMO_EXPR_
                |  .
FATOR_EXPR      -> "(" EXPR_CHAMADA ")"
                |  PARM_CALL .
ADDOP_EXPR      -> "+"  |  "-" .
MULOP_EXPR      -> "*"  |  "/" .
PARM_CALL       -> INDIREC_VAR
                |  ENDER_VAR
                |  VAR_FUNC_PARM
                |  INTEIRO
                |  FLOAT
                |  CADEIA .
INDIREC_VAR     -> "*" INDIREC_VAR_ .
INDIREC_VAR_    -> INDIREC_VAR
                |  IDENT .
VAR_FUNC_PARM   -> IDENT PARM_OPCIONAL .
PARM_OPCIONAL   -> PARM_CHAMADA
                |  .
```

Num sistema aplicativo comercial, podemos imaginar a necessidade de informar ao usuário quantos registros de um determinado arquivo de dados já foram processados. A apresentação atualizada desta informação deve ser efetuada após a leitura de um novo registro. A definição do "*lay-out*" da apresentação, utilizando-se a seção de definição dos tokens de saída, pode ser feito como segue :

```
saida
  Reg_Processados(int num; int num_total) ->
                  escreva(1,50,"Foram processados <")
                  escreva(1,70,num)
                  escreva(1,72,"> de <")
                  escreva(1,77,num)
                  escreva(1,79,">")
```

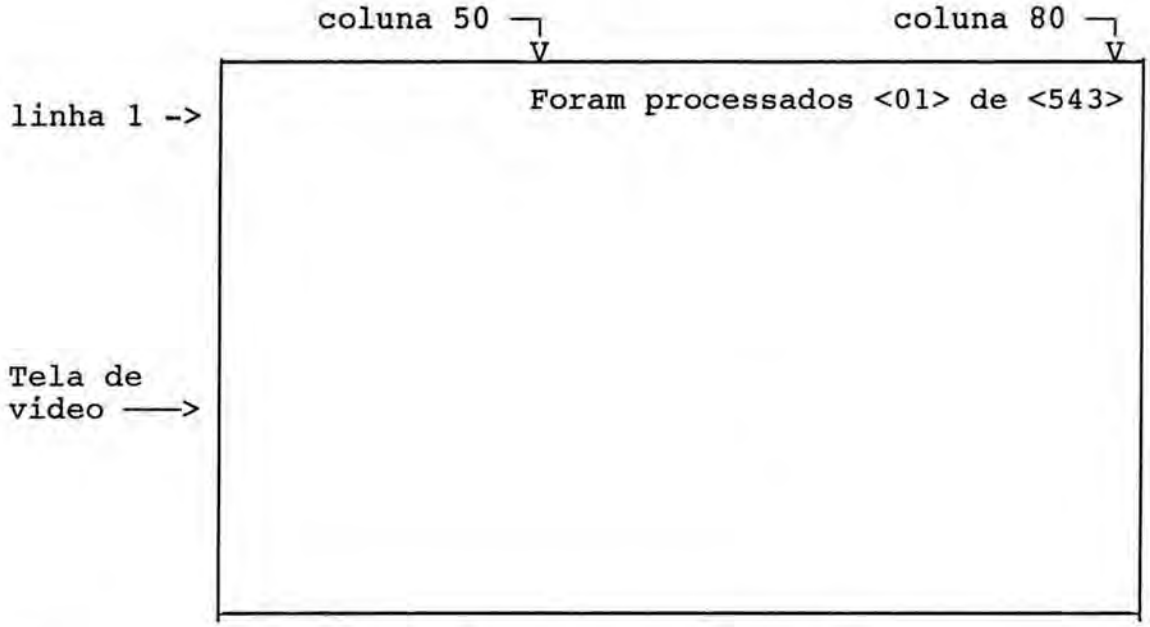

Figura 5.1 Apresentação de um token de saida

5.7 Seção de Regras de Associação

Esta seção mapeia as relações existentes entre os elementos declarados na seção de geração (menus, formulários, linguagens de comando e tokens de saída) e as produções definidas na seção sintaxe abstrata&semântica (ver 5.8). Estas relações definem os pontos de interação na estrutura do diálogo. Um ponto de interação identifica uma produção que necessita da execução de uma interação com o usuário antes de sua redução. Cada ponto de interação pode ser visto como uma declaração de pré-condição para a produção correspondente. A identificação de uma produção

deve ser única para garantir que não ocorram ambigüidades na sua localização.

Como foi exposto na seção de geração, cada elemento ali declarado pode utilizar um conjunto de atributos. Estes atributos são herdados das produções com as quais estão relacionados ou estão definidas como constantes nas regras de associação. O mecanismo utilizado que define esta interligação é composto pela declaração dos atributos nos corpos das regras de associação, de forma que visualmente lembrem chamadas de funções. O reconhecedor de GRAEDIUS realiza a busca dos valores dos atributos nas produções e os associa, de forma posicional, aos atributos dos elementos de geração correspondentes.

As regras de mapeamento dos elementos utilizados como exemplos na seção de geração poderiam ser assim expressas :

```
regras de associação
                                          /* regras de : */
  inicio -> Menu0;                        /*  entrada */
  inc    -> Ficha_Cadastral(local_x, local_y);
                                          /*  entrada */
  util   -> Linguagem_01(texto);          /*  entrada */
  casa   -> Des_Casa(pos_x, pos_y);       /*  saida   */
```

Nestes exemplos os elementos de geração de entrada "Ficha_Cadastral" e "Linguagem_01" herdam atributos de todas as produções que possuem em seu corpo referencias às produções identificadas pelos rótulos "inc" e "util", e o elemento de geração de saída "Des_Casa" herda atributos da produção rotulada por "casa".

Assim como os elementos de geração são classificados como sendo de entrada e de saída, assim também as regras de associação podem ser de entrada ou de saída. A diferenciação entre as regras que fazem associações de entrada e de saída é feita pela natureza dos elementos associados pela regra. Se uma regra associa uma produção a um elemento declarado no grupo de entrada da seção de geração, então é uma regra de entrada. As regras de associação de saída relacionam produções a elementos declarados no grupo de saída da seção de geração.

No exemplo apresentado a produção rotulada como "inicio" está associada com o elemento de geração de entrada (menu) "Menu0", portanto é uma regra de entrada. Já a produção rotulada como "casa" está associada com o elemento de geração de saída "Des_Casa".

As interações descritas pelos elementos de geração de entrada são executadas antes do uso de corpo da produção, a qual está associado, num reconhecimento. A execução dos elementos associados nas regras de entrada é uma pré-condição da produção associada.

A execução dos elementos de geração identificados pelas regras de saída se dá quando do total reconhecimento da produção a qual está associado. A execução dos elementos das regras de saída é uma pós-condição da produção associada.

A sintaxe utilizada para descrever o mapeamento e o intercâmbio de atributos é descrita pela gramática :

```
REGRAS_ASSOC    -> "regras" ID_VAR_REGRA CORPO_REGRA
                   FIM_SECCAO .
ID_VAR_REGRA    -> "de" "associacao"
                |  "associacao"
                |  .
CORPO_REGRA     -> PROD_ASSOC CORPO_REGRA_ .
CORPO_REGRA_    -> CORPO_REGRA
                |  .
PROD_ASSOC      -> IDENT "->" IDENT PARM_OPCIONAL ";" .
PARM_OPCIONAL   -> PARM_CHAMADA
                |  .
PARM_CHAMADA    -> "(" LST_PARM_CALL ")" .
LST_PARM_CALL   -> EXPR_CHAMADA LST_PARM_CALL_
                |  .
LST_PARM_CALL_  -> "," LST_PARM_CALL
                |  .
EXPR_CHAMADA    -> TERMO_EXPR EXPR_CHAMADA_ .
EXPR_CHAMADA_   -> ADDOP_EXPR TERMO_EXPR EXPR_CHAMADA_
                |  .
TERMO_EXPR      -> FATOR_EXPR TERMO_EXPR_ .
TERMO_EXPR_     -> MULOP_EXPR FATOR_EXPR TERMO_EXPR_
                |  .
FATOR_EXPR      -> "(" EXPR_CHAMADA ")"
                |  PARM_CALL .
ADDOP_EXPR      -> "+" |  "-" .
MULOP_EXPR      -> "*" |  "/" .
PARM_CALL       -> INDIREC_VAR
                |  ENDER_VAR
                |  VAR_FUNC_PARM
                |  INTEIRO
                |  FLOAT
                |  CADEIA .
INDIREC_VAR     -> "*" INDIREC_VAR_ .
INDIREC_VAR_    -> INDIREC_VAR
                |  IDENT .
VAR_FUNC_PARM   -> IDENT PARM_OPCIONAL .
```

5.8 Seção de Sintaxe Abstrata & Semântica

Esta seção representa o "coração" da GRAEDIUS, pois é onde são descritas a estrutura do diálogo da IU e os pontos de ligação entre a IU e a aplicação. É nesta seção que se delimitam as responsabilidades das funções da aplicação e as responsabilidades do gerenciador da IU.

É composta por uma cláusula de definição de atributos/equações semânticas globais e um conjunto de produções. Estas são compostas por :

- um conjunto de rótulos de identificação das produções;
- declarações dos atributos das produções; e
- um conjunto de equações de atributos da gramática.

5.8.1 Área Global

A área de definição de comandos globais tem por objetivo permitir a especificação de procedimentos que sejam globais à aplicação. Nesta área também são informadas as estruturas de dados globais da IU.

5.8.2 Produções

A gramática que define a estrutura do diálogo deve ser uma GLC sem recursão a esquerda. Uma produção que defina um ponto de interação deve permitir que os elementos de geração de entrada sejam tratados como pré-condição pelo reconhecedor da GRAEDIUS. A execução das representações dos tokens de saída deve ser executada como uma pós-condição da produção, portanto todas as informações necessárias à execução da representação devem estar disponíveis no final do reconhecimento da produção.

Esta gramática define qual a estrutura do diálogo que estará à disposição do usuário. Para a definição de estruturas de diálogos planos ou não

hierárquicos (redes) (ver 3.1.1), devem ser criadas produções *dummy* [GRE 86]. Estas produções tem como única função permitir ao reconhecedor de GRAEDIUS a correta simulação destes tipos alternativos de diálogos. GRAEDIUS permite somente a definição e manipulação de diálogos tipicamente hierárquicos. Esta limitação é originada na própria natureza de uma gramática que é hierárquica.

5.8.2.1 Rótulos

Os rótulos usados na identificação das produções permitem que a um mesmo nome de produção seja associado mais de um estilo de interação de entrada e saída. Esta capacidade permite que a partir de um nodo da estrutura do diálogo estejam disponíveis vários estilos de interação, utilizados de acordo com as tarefas selecionadas pelo usuário. O que determina qual estilo será utilizado é o valor do token que vai ser reconhecido. O token indica qual das produções será utilizada; assim o estilo fica definido pela regra de associação.

O trecho de definição de uma IU imaginária exemplificada a seguir define os rótulos "r1" e "r2" para a produção "p1". O rótulo "r1" identifica a primeira ocorrência de "p1", enquanto que "r2" a segunda ocorrência. As regras de associação mapeiam para a produção identificada por "r1" o uso do elemento de geração "Menu01" e para "r2" o elemento "LC01".

```
      .
      .
      .
regras
   r1 -> Menu01;
   r2 -> LC01;
```

```
        .
        .
        .
sintaxe abstrata&semantica
        .
        .
        .
     P  -> P1 P2.
[r1] P1 -> xxx P11.
[r2] P1 -> yyy P12.
        .
        .
        .
```

5.8.2.2 Declaração dos Atributos

Os atributos manipulados por GRAEDIUS são "tipados", isto é, são de um tipo bem definido e imutável. A declaração dos tipos dos atributos tem como objetivo facilitar a avaliação das equações. O uso da "tipagem" permite que sejam feitas checagens da semântica estática nas equações. GRAEDIUS manipula três tipos básicos de atributos :

- os locais;
- os herdados; e
- os sintetizados.

Um atributo local é aquele declarado na produção e acessível somente em seu corpo.

Os atributos herdados são os atributos acessíveis aos elementos da produção que vai ser reduzida. Indicam quais atributos da produção serão manipulados como atributos estrangeiros pela produção a ser reduzida.

Um atributo sintetizado é um atributo local que vai ter seu valor atualizado, manipulado ou determinado em equações semânticas de produções derivadas da produção de origem. O mecanismo utilizado por GRAEDIUS para a manipulação destes tipos de atributos é semelhante ao utilizado nas linguagens de programação "C" [BOR 88a] e "PASCAL" [BOR 88b] para o tratamento de variáveis locais e estrangeiras.

```
        .
        .
        .
1>P1
2>LOCAL    : { int ret;          /* atributo sintetizado*/
3>             int num = 15;     /* atributo sintetizado*/
4>             int aux;}         /* atributo local      */
5>ACOES_IN : { aux = 20; }
6>     -> xxx
7>        P11(&num,ret)
8>           {wprintf("%d",ret);}
9>        P1
10>     |  "yyy".
11>
12>P11(int *num; int retorno) /* num será herdado e sint.
13>                             retorno será sintet.  */
14>ACOES_OUT : { if *num > 10
15>               {
16>                retorno = 0;
17>                Poe_Buffer("yyy");
18>               }
19>               else
20>               {
21>                *num = 0;
22>                retorno = 1;
23>                Poe_Buffer("xxx");
24>               }
25>             }
26>     -> "www" .
        .
        .
        .
```

O exemplo acima apresenta um trecho da seção de sintaxe abstrata&semântica. As linhas numeradas de 02 a 04 declaram os atributos locais *ret*, *num* e *aux*. O atributo *num* é inicializado com o valor 15, esta é uma facilidade conferida pelo reconhecedor de GRAEDIUS. O não terminal

P11, no corpo da produção **P1**, herdará os atributos *num* e *ret*. O atributo *num*, além de herdado será também sintetizado no corpo da produção **P11** (linhas 12 à 26) pelas suas ações semânticas.

5.8.2.3 Cláusula Ações_In

Comandos que devem ser executados antes do uso do corpo da produção para fatoração são declarados na cláusula "ACOES_IN" da produção. Determinam que ações são comuns a todos os corpos alternativos daquela produção e que devem ser executadas antes da redução da produção;

No exemplo apresentado anteriormente, a cláusula de entrada da produção **P1** (linha 5) atribui o valor 20 ao atributo *aux*. Somente será executada se o terminal "*xxx*" ou "*yyy*" for reconhecido.

5.8.2.4 Cláusula Ações_out

Comandos que devem ser executados quando da completa redução do corpo da produção são declaradas na cláusula "ACOES_OUT" da produção. Determinam que ações são comuns a todos os corpos alternativos da produção e que devem ser executadas depois da redução do corpo utilizado.

No exemplo usado em 5.8.2.2, a cláusula de saída está associada à produção **P11**, e está declarada das linhas 14 à 25. Sua execução testará se o atributo herdado *num* é maior do que 10. Se for, então atribuirá o valor zero ao atributo *retorno*, caso contrário atribuirá o valor um ao

atributo retorno e zero ao atributo *num*. Somente será executada se o terminal "*www*" for reconhecido.

5.8.2.5 Ações Semânticas

As ações semânticas devem ser executadas quando um determinado elemento do corpo da produção é reduzido/reconhecido. Determinam o processamento que deve ser feito sobre os atributos locais da produção, quando da redução de um não-terminal ou do reconhecimento de um terminal. Seu uso permite a especificação das computações intermediárias de uma IU, bem como permite que a IU controle e execute as computações que dizem respeito somente aos seus elementos. Um exemplo claro deste tipo de computação é o cálculo para reposicionar um ícone, num estilo de comando direto, quando selecionado e movimentado pela tela.

No exemplo usado em 5.8.2.2, as ações semânticas são compostas pelas cláusulas de entrada, saída e pelos comandos declarados entre chaves ({ e }) nas linhas 5, 8 e 14 à 25. No caso de ser necessária a alteração do fluxo de reconhecimento, tokens podem ser inseridos na memória do reconhecedor léxico através do uso da função **POE_BUFFER**, conforme mostram as linhas 17 e 23. A primeira chamada inclui o token "*yyy*" e a segunda o token "*xxx*". Um destes será o próximo token a ser processado (reconhecido).

5.8.3 Definição Gramatical

A sintaxe que define o formato da definição da **sintaxe abstrata&semântica** é dado pela gramática que

segue :

```
ABST_SEMAN      -> "sintaxe" ID_VAR_A_S CLAUSULA_GLOBAL
                   CORPO_A_S FIM_SECCAO .
ID_VAR_A_S      -> "abstrata" "&" "semantica"
                |  AeS
                |  .
CLAUSULA_GLOBAL-> "global" ACAO_SEMAN
                |  .
CORPO_A_S       -> PRODUCAO CORPO_A_S_ .
CORPO_A_S_      -> CORPO_A_S
                |  .
PRODUCAO        -> ID_PROD IDENT DECL_PARM LOCAL_PROD
                   ACOES_IN_PROD ACOES_OUT_PROD
                   "->" CORPO_SINT ";" .
ID_PROD         -> "[" IDENT "]"
                |  .
DECL_PARM       -> "(" LST_PARM ")"
                |  .
LST_PARM        -> PARAMETRO LST_PARM_
                |  .
LST_PARM_       -> "," PARAMETRO
                |  .
PARAMETRO       -> CL_CONST CL_MOD_SINAL CL_MOD_TAM CL_TIPO
                   POINTER NOME_PARM LST_PARM_
                |  "..." .
CL_CONST        -> "const"
                |  .
CL_MOD_SINAL    -> "signed"  |  "unsigned"  |  .
CL_MOD_TAM      -> "short"   |  "long"      |  .
CL_TIPO         -> "char"    |  "int"       |  "void"
                |  "float"   |  "double"    |  IDENT .
POINTER         -> CL_MOD_POINTER "*" POINTER_
                |  .
POINTER_        -> "*" POINTER_
                |  .
CL_MOD_POINTER -> "far"    |  "near"     |  "huge"   |  .
NOME_PARM       -> IDENT
                |  .
LOCAL_PROD      -> "local"   ":" ACAO_SEMAN
                |  .
ACOES_IN_PROD   -> "acoes_in"  ":" ACAO_SEMAN
                |  .
ACOES_OUT_PROD -> "acoes_out" ":" ACAO_SEMAN
                |  .
CORPO_SINT      -> CORPO_PROD CORPO_SINT_ .
CORPO_SINT_     -> "|" CORPO_SINT
                |  .
CORPO_PROD      -> IDENT PARM_OPCIONAL CORPO_PROD
                |  CADEIA CORPO_PROD
```

```
                  |  ACAO_SEMAN CORPO_PROD
                  |  .
PARM_OPCIONAL     -> "(" LST_PARM ")"
                  |  .
```

6 PROJETANDO UMA IU

Neste capitulo é apresentada uma proposta de metodologia de projeto de IU, composta por um conjunto de 11 etapas. Estas etapas estão direcionadas para o uso da proposta GRAEDIUS. As etapas 1 a 8 são comuns a outras metodologias de projeto de IUs, e as etapas 9 a 11 são especificas ao uso da GRAEDIUS. Na seção 6.1 apresenta um sistema aplicativo que será utilizado na exemplificação de cada uma destas etapas. Pelo seu cunho didático, este sistema não apresenta, nem complexidade, nem sofisticação nas tarefas oferecidas. O sistema busca somente explorar as potencialidades da proposta. A seção 6.2 apresenta e descreve suscintamente as etapas da metodologia de projeto de IU mencionadas. As seções 6.3 até 6.12 fazem uma descrição mais detalhada das tarefas executadas em cada etapa, ilustrando-as com o detalhamento do sistema apresentado na seção 6.1. Por fim, a seção 6.13 apresenta o aspecto visual da IU obtida.

6.1 Sistema de Exemplo

O problema que este sistema deverá solucionar está relacionado com as atividades de edição de figuras gráficas. As figuras são simples e formadas pela composição de circulos, retas, pontos e texto. Para auxiliar o trabalho do usuário, é interessante que disponha de facilidades de armazenamento, recuperação e alteração das figuras já criadas. Um exemplo simples de figuras que se deseja manipular é mostrado na figura 6.1.

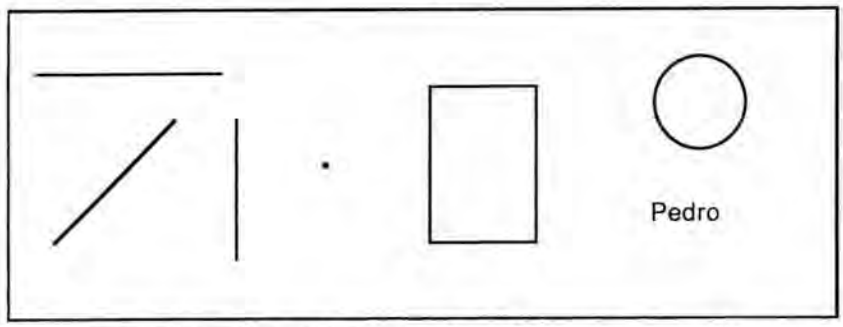

Figura 6.1 Exemplo da classe de figuras manipuladas pelo sistema proposto

A utilidade ou factibilidade de uso deste sistema é questionável, mas atinge seu objetivo que é permitir a ilustração de todas as etapas necessárias para o desenvolvimento de uma boa IU.

6.2 Etapas do Projeto da IU

A proposta de etapas aqui apresentada não foi encontrada em nenhuma referência consultada, mas derivou-se a partir das experiências que o autor acumulou na confecção : a) de sistemas computacionais, b) dos exemplos apresentados neste trabalho e c) do protótipo que implementa GRAEDIUS. Como já mencionado na introdução deste capítulo, a metodologia aqui proposta somente é apresentada como um exemplo.

No projeto de um sistema, é interessante que algumas etapas sejam observadas para a obtenção de uma boa IU. A figura 6.2 apresenta um diagrama de um DFD (Diagrama de Fluxo de Dados) [GAN 83] que ilustra as interdependências existentes entre as etapas. As etapas estão enumeradas de 1 a 11 e são melhor discutidas nas seções que seguem. O anexo 1 apresenta uma tabela que resume as principais atividades a serem desenvolvidas em cada etapa.

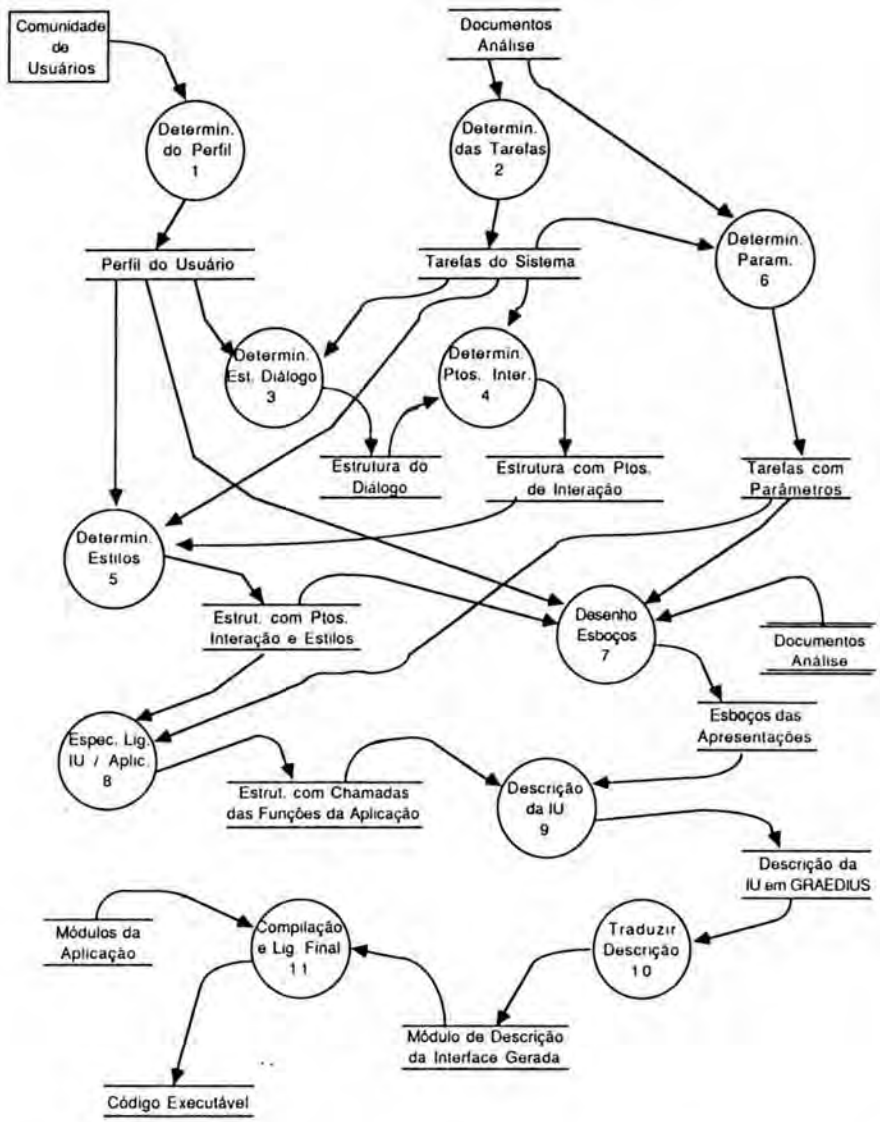

Figura 6.2 DFD do paralelismo das etapas de uso da GRAEDIUS

A compilação desta metodologia baseou-se nas metodologias existentes para a análise de sistemas. A tabela 6.1 compara a metodologia proposta com a metodologia apresentada por Gane/Sarson em [GAN 83]. São traçadas as equivalências entre as etapas das duas metodologias.

Tabela 6.1 correspondências entre metodologias

proposto	análise estruturada de Gane
det. do perfil det. tarefas aplic. det. parm. tarefas	levantamento de requisitos
det. estrut. diálogo det. pto. interação	projeto lógico funcional (DFD)
det. est. interação	projeto lógico da estrut. dados
desenho de esboços	projeto físico das estrut. dados
espec. lig. IU/aplic.	projeto físico do sistema
descr. da IU	elaboração do pseudocódigo
traduzir descrição	programação do sistema
comp. e lig. final	integração e implementação

6.3 Perfil Médio dos Usuários

Nesta etapa, são levantados os tipos físicos, psicológicos, ergonômicos, sociais, culturais e intelectuais médios da comunidade de usuários do sistema. Este levantamento vai possibilitar um melhor projeto da estrutura de diálogo e das técnicas de interação postas a disposição da comunidade de usuários. Esta determinação permite projetar um diálogo mais compatível com as espectativas dos usuários [SHN 86].

Os fatores sociais, culturais e intelectuais da comunidade possuem reflexo sobre os fatores relacionados com :

- tempo de aprendizado;

- velocidade de execução das tarefas;
- média de erros por usuário;
- satisfação subjetiva; e
- facilidade de memorização do uso do sistema.

Estes fatores são somente alguns apontados por Shneiderman em [SHN 86]. Fischer [FIS 88] diz que a importância da determinação do perfil da comunidade permite que o projeto de sistemas seja baseado em dois modelos de interação : a) o para usuários sofisticados e b) o para usuários ocasionais.

A comunidade de usuários do sistema exemplo é composta por pessoas com grande capacidade de aprendizado, mas sem nenhuma experiência computacional. Seus niveis sociais, culturais e intelectuais possuem um valor 8 numa escala de 10. Porém seus contactos com o computador serão ocasionais.

6.4 Tarefas Disponiveis

A determinação das tarefas executadas pelo sistema em estudo é fator de grande importância no processo de projeto dos estilos de interação oferecidos. O conjunto de tarefas, disponiveis à comunidade de usuários, é facilmente determinado de posse do projeto lógico do sistema em estudo. Normalmente, as tarefas são identificadas pelas funções que implementam/manipulam a troca de informações entre o sistema e seu mundo externo. Num sistema projetado com o auxilio da análise estruturada de Gane/Sarson [GAN 83], as

tarefas normalmente são coincidentes com os estímulos que o sistema recebe do mundo externo, e é facilmente determinado no nível zero.

No sistema exemplo, o nível zero determina que sejam enumeradas como tarefas do usuário as funções que permitam-lhe :

- criar arquivos de desenho;
- alterar arquivos de desenho;
- salvar arquivos de desenho novos/alterados;
- excluir arquivos de desenho;
- incluir primitivas gráficas;
- excluir primitivas gráficas;
- mover primitivas gráficas;

6.5 Estrutura do Diálogo

A estrutura de diálogo pode ter vários formatos de organização (ver 3), mas devem sempre levar em consideração as tarefas disponíveis aos usuários. A estrutura de diálogo determina qual a metáfora de diálogo que será adotada para o sistema em estudo. Sua determinação deve levar em consideração o perfil levantado na etapa 1. Esta estrutura mapeia a seqüência de ações que o usuário deve executar para selecionar a execução das tarefas.

Fisher em [FIS 88] sugere que o modelo de interação para usuários sofisticados deve ser adotado se há

um uso por tempo prolongado, quando o uso é freqüente e quando a comunidade teve um treinamento expressivo no seu uso. De outra forma, o usuário é convidado a "dirigir" a execução do sistema. Já o modelo para usuários ocasionais deve ser adotado quando o tempo de uso for pequeno, quando o uso for ocasional e quando os usuários forem casuais e não tiveram muito treinamento. De outra forma, o usuário é convidado a "sentar-se" no assento de passageiros e o sistema "dirige" a execução. As figuras 6.3 e 6.4 apresentam uma representação esquemática dos modelos propostos por Fisher.

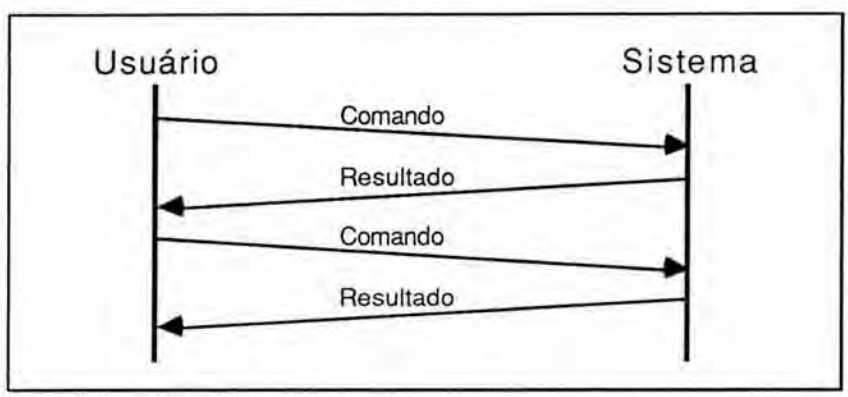

Figura 6.3 O modelo de interação para usuários sofisticados

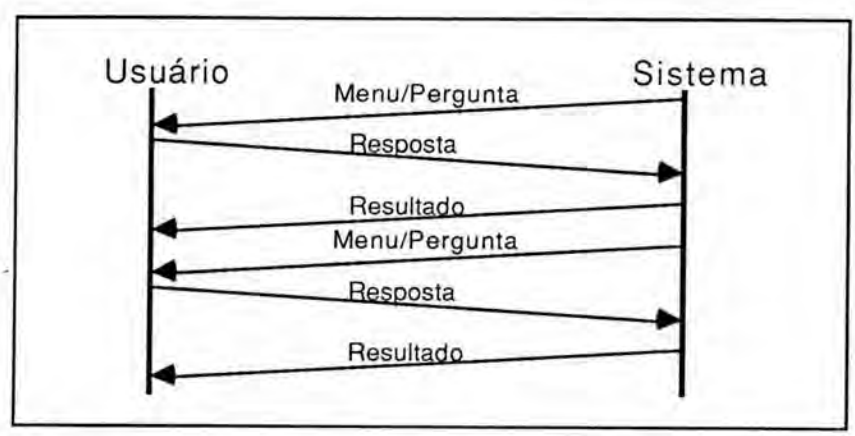

Figura 6.4 O modelo de interação para usuários ocasionais

Com pequenas variações, estes modelos são adotados em bom número de bibliografias consultadas [COU 85] [SHN 86] [FIS 88] [MCC 89]. McClure apresenta em [MCC 89] na tabela 6.1 as diferentes espectativas de informação e atividades que o sistema deve possuir. Esta tabela em conjunção com o levantamento do perfil, permitem um bom enquadramento do usuário e suas necessidades, direcionando razoavelmente o projeto da estrutura do diálogo.

Tabela 6.1 tipo de usuário X espectativas

tipo	espectativas
Mestre	- capaz de expandir o sistema para que suporte novas funções e procedimentos. - necessita poucas explicações e mensagens do sistema.
Especialista	- utiliza o sistema com facilidade. - necessita de atalhos para as opções e informações de "status".
Intermediário	- é capaz de entender conceitos de alto nível e trabalha ao nível de tarefas. - pode tomar a iniciativa, pois sabe como traduzir tarefas em seqüências de comandos que o sistema deve executar.
Novato	- é capaz de entender conceitos de baixo nível e trabalhar ao nível de comandos. - necessita de respostas do sistema.
Inexperiente	- opera no nivel de caracteres simples. - necessita de exemplos para basear seu trabalho. - necessita orientação e guias de navegação e respostas constantes do sistema.

A estrutura de diálogo adotada no sistema exemplo é o apresentado na figura 6.5. Adotou-se a estrutura hierárquica, baseando-se no perfil levantado, ou seja, os usuários são ocasionais, e nesta caso este tipo de estrutura é o mais indicado [SHN 86]. Caso GRAEDIUS oferecesse capacidade de modelagem, o estilo assincrono

seria boa opção, pois o número de tarefas é pequeno e facilmente representado na forma gráfica (ícones).

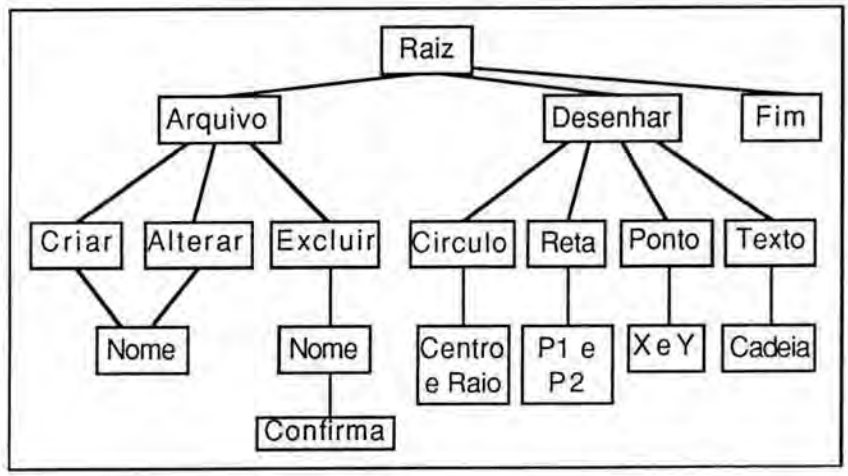

Figura 6.5 A estrutura do diálogo do sistema de exemplo

6.6 Pontos de Interação

A definição dos pontos de interação é feita diretamente sobre a estrutura do diálogo adotada. A determinação dos pontos de interação de entrada indica em quais pontos da estrutura do diálogo o gerenciador da IU deve solicitar ao usuário uma intervenção. Em cada uma destas intervenções o usuário informa qual a próxima "direção" que deve ser seguida pelo gerenciador sobre a estrutura do diálogo. Quando o gerenciador atinge algum nodo terminal da estrutura, então a determinação da tarefa está completa e pode ser executada.

A figura 6.6 acrescenta à estrutura do diálogo os pontos de interação do sistema. Nestes pontos, a interação pode tanto ser de entrada como de saída. No caso de um ponto definir uma interação de saída, o que se determina é quando as representações associadas aos retornos semânticos

dos processamentos das intervenções de entrada devem ser exibidos.

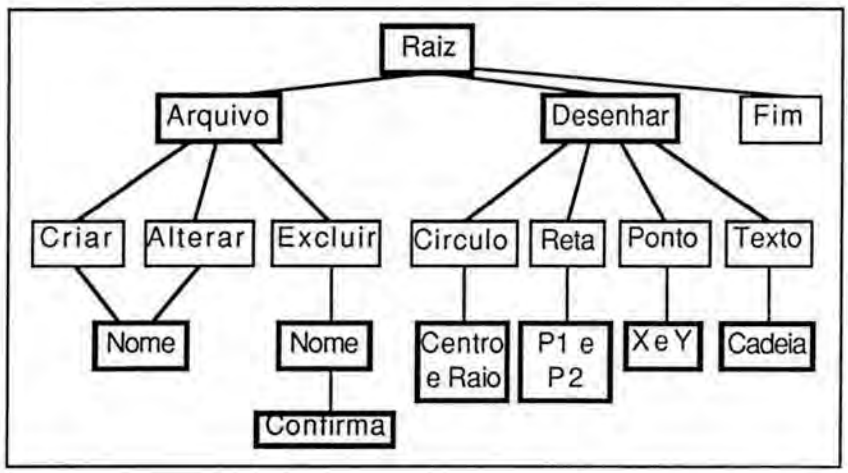

Figura 6.6 Pontos de interação marcados sobre a estrutura do diálogo do sistema de exemplo

6.7 Estilos de Interação

Com base no perfil médio da comunidade de usuários, é possível realizar a seleção das técnicas de interação que melhor se adaptam à comunidade. O objetivo básico destas seleções é encontrar um equilíbrio harmônico entre as necessidades/potencialidades dos usuários e as capacidades oferecidas pelos estilos de interação a serem adotados.

No projeto de uma IU deve-se buscar a utilização de estilos de interação que facilitem a compreensão dos conceitos utilizados e das tarefas disponíveis, sendo totalmente desaconselhado o uso de estilos que, para a comunidade alvo, tornem a compreensão complexa e difícil

[MCC 89]. A figura 6.7 apresenta alguns conceitos que facilitam e dificultam a vida do usuário.

Figura 6.7 Conceitos fáceis e difíceis de compreender

Fisher indica o uso de estilos como menus, formulários para usuários ocasionais e linguagens de comando para usuários sofisticados [FIS 88]. A tabela 6.2 mostra os prós e contras dos estilos de interação de menus e linguagens de comando melhor descritos no capítulo 3.

Tabela 6.2 prós e contras dos estilos de menus e linguagens de comando

estilo	vantagens	desvantagens
Ling. Comando	muito eficiente interações ricas e expressivas	difícil aprender se não usado é fácil de esquecer
Menus	fáceis de usar fácil de aprender	interação lenta deve-se saber o que fazer

A tabela 6.3 enumera os estilos de interação de entrada selecionados para o sistema exemplo. Para cada um dos pontos de interação é indicado o estilo selecionado. Como pode ser constatado, uma combinação de estilos é sugerida. Esta mescla permite um rápido aprendizado e um bom conforto na utilização do sistema, além de ser ideal à usuários ocasionais.

Tabela 6.3 pontos de interação X estilos de interação

ponto	estilo
Raiz	menu
Arquivo	menu
Desenhar	menu
Nome 1	formulário
Nome 2	formulário
Continua	formulário
Centro e Raio	formulário
P1 e P2	formulário
X e Y	formulário
Cadeia	formulário

6.8 Parâmetros das Tarefas

Uma vez que os estilos de interação necessários em cada ponto de interação estão determinados, torna-se necessário o conhecimento das quantidades e características de cada um dos parâmetros utilizados para a completa definição das tarefas a serem executadas em cada um destes pontos. Novamente torna-se necessário consultar as informações contidas nos documentos gerados pela análise lógica do sistema. Se foi utilizada a análise estruturada, então uma consulta ao dicionário de dados permite a

determinação de todas as informações que se fazem necessárias para o reconhecimento e processamento das tarefas disponiveis no sistema. Estas informações compoem, justamente, os parâmetros que as tarefas precisam para serem completamente especificadas.

Tabela 6.4 funções X parâmetros

nome	parâmetros	tp	tam
Abre_arq	nome	a	12
Alt_arq	nome	a	12
Salva_arq	nome	a	12
Exclui_arq	nome	a	12
Des_circulo	x_centro	n	3
	y_centro	n	3
	raio	n	3
Des_reta	x_p1	n	3
	y_p1	n	3
	x_p2	n	3
	y_p2	n	3
Des_pto	x_ponto	n	3
	y_ponto	n	3
Inc_texto	x_inicio	n	3
	y_inicio	n	3
	cadeia	a	15

De posse da relação destes parâmetros é necessário determinar o local de origem de cada um deles. Se é originado por outro sistema externo, como é feita sua identificação; se é fornecido pelo usuário, qual o tipo, tamanho, dominio, etc. Quando este trabalho estiver concluido, torna-se fácil a determinação dos parâmetros que

o usuário deverá digitar para especificar completamente uma tarefa.

Os parâmetros necessários para cada uma das tarefas do sistema exemplo, relacionadas na seção 6.4, possuem a especificação de parâmetros apresentada na tabela 6.4.

6.9 Esboços das Apresentações

A necessidade destes esboços é algo que pode ser questionado, mas em termos documentacionais são indispensáveis e são eles que permitem uma validação parcial da IU junto ao usuário final. Sua utilidade é sentida em dois momentos distintos : 1) na apresentação e discussão preliminar da IU com a comunidade de usuários e 2) no momento da descrição textual da IU em GRAEDIUS.

Os esboços das apresentações de um projeto de uma IU devem incluir não só informações das apresentações dos estilos de interação de entrada, mas também devem permitir a visualização das saidas e das áreas de trabalho do sistema. Para os estilos de interação de entrada devem facilitar a compreensão das diferentes combinações de estilos disponíveis, assim como a forma de informação dos parâmetros das tarefas, enfim, deve permitir a simulação da IU.

Para o sistema exemplo, tendo por base os estilos de interação especificados e os parâmetros determinados, foram projetados os esboços apresentados nas figuras 6.8

até a figura 6.16. As figuras 6.9 à 6.11 apresentam os croquis dos menus da aplicação. Estes, serão utilizados para permitir ao usuário selecionar os comandos desejados. Das figuras 6.12 até a figura 6.16 apresentam os formulários usados. Estes são empregados, exclusivamente, para a informação dos parâmetros das tarefas selecionadas.

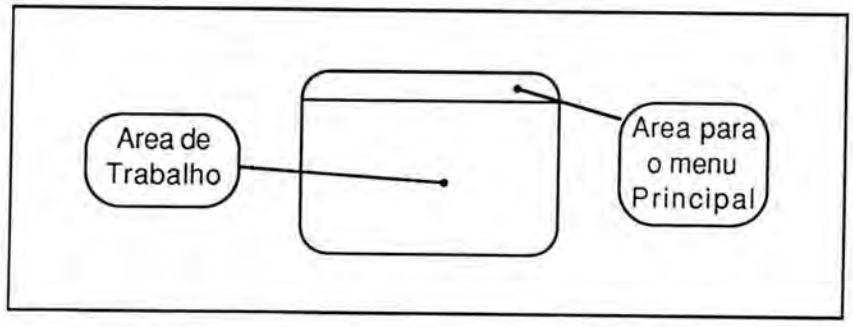

Figura 6.8 Área de trabalho

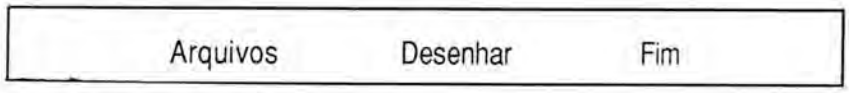

Figura 6.9 Menu principal

Figura 6.10 Menu de arquivos

Figura 6.11 Menu de primitivas

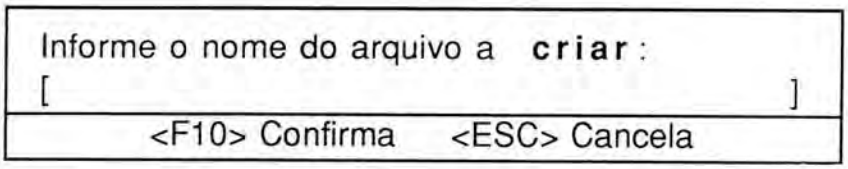

Figura 6.12 Formulário para informação do nome do arquivo a criar

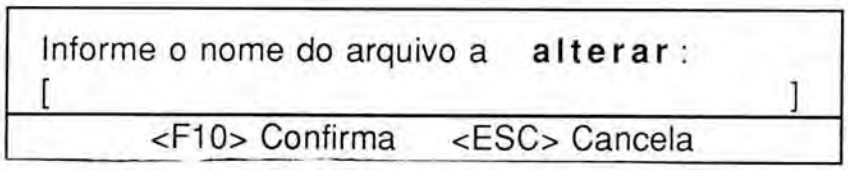

Figura 6.13 Formulário para informação do nome do arquivo a alterar

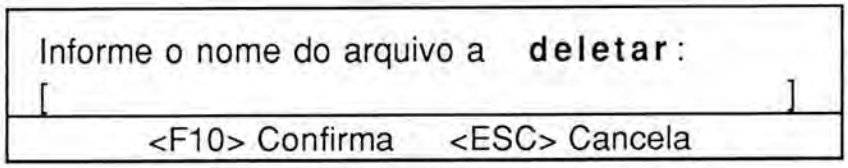

Figura 6.14 Formulário para informação do nome do arquivo a excluir

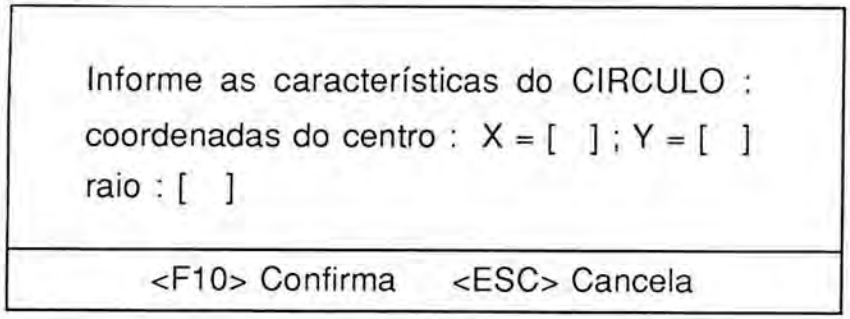

Figura 6.15 Formulário para informar o raio e as coordenadas do centro de um circulo

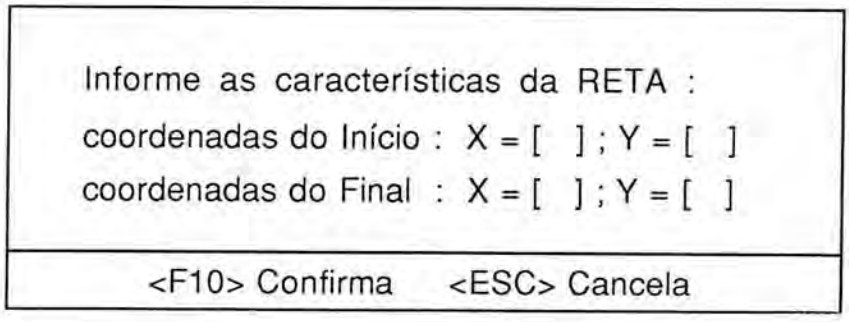

Figura 6.16 Formulário para informar as coordenadas dos extremos de uma reta

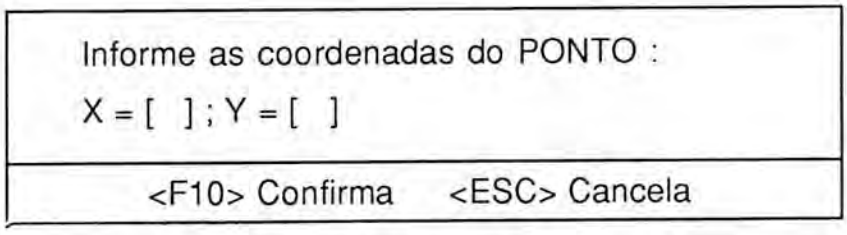

Figura 6.17 Formulário para informar as coordenadas de um ponto

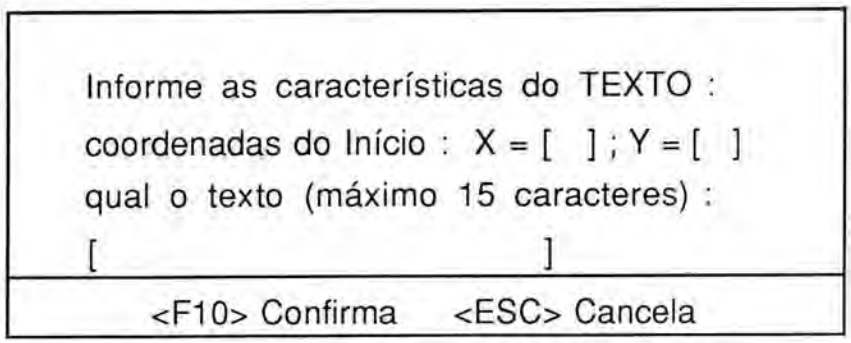

Figura 6.18 Formulário para informar as coordenadas e o texto a mostrar

6.10 Ligações entre a IU e a Aplicação

As ligações são criadas nos nodos da estrutura do diálogo. Nos nodos terminais são invocadas as funções que executam as tarefas especificadas pelos usuários e nos nodos intermediários, são invocadas as funções responsáveis pelas computações auxiliares. Uma computação auxiliar pode ser exemplificada com uma função que, verificando uma determinada condição, habilita a execução de determinado conjunto de tarefas dependentes desta condição. A execução das funções associadas é feita com chamadas declaradas nas ações semânticas das produções da gramática que descreve a estrutura do diálogo.

A tabela 6.5 mostra as funções do sistema que serão executadas no reconhecimento de cada tarefa especificada na seção 6.4.

Tabela 6.5 tarefas do usuário X funções do sistema

tarefa	função
criar arquivo	Abre_Arq
alterar arquivo	Altera_Arq
salvar arquivo	Salva_Arq
excluir arquivo	Del_Arq
incluir circulo	Des_Circulo
incluir reta	Des_Reta
incluir ponto	Des_Pto
incluir texto	Inc_Texto

6.11 Descrição da IU

Tendo em mãos a estrutura do diálogo, todos os esboços da apresentação, o nome das funções que serão invocadas para a execução de cada tarefa e a relação dos processamentos intermediários necessários, pode-se fazer a descrição da IU usando a linguagem definida pela proposta GRAEDIUS. A maneira de descrever cada um dos componentes da IU é a apresentada no capitulo 5 das seções 5.3 até a seção 5.8. O anexo 8 lista a descrição da IU definida para o sistema de exemplo.

6.12 Compilação da Descrição

Sendo um SDIU, GRAEDIUS compila um programa fonte, escrito em linguagem C, a partir da descrição da IU. Este programa pode ser compilado em separado dos outros módulos do sistema. As implementações das funções que implementam as tarefas do sistema é feita em módulo(s) separado(s). No tempo de ligação, este módulo é juntado com os demais. A criação de um programa em linguagem intermediária permite que pequenas alterações na IU sejam

feitas diretamente sobre o código, ainda que isto possa gerar inconsistências na documentação associada.

Exemplificar esta etapa é algo difícil, pois as ações do projetista limitam-se a comandar o início da tradução.

6.13 Execução

Na execução do código binário gerado no último passo do projeto da IU, obtem-se uma apresentação visual conforme o mostrado nas figuras de 6.19 a 6.30.

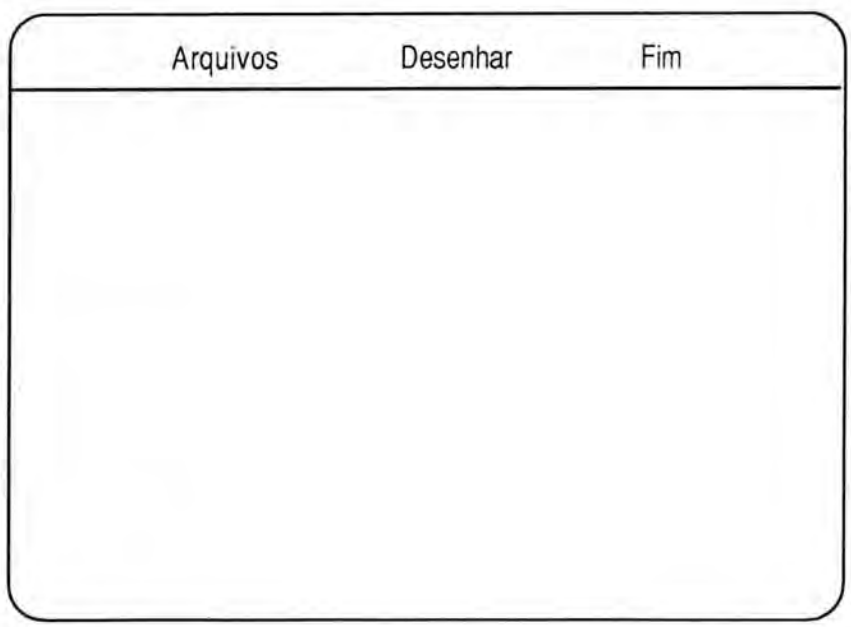

Figura 6.19 Tela de apresentação do sistema

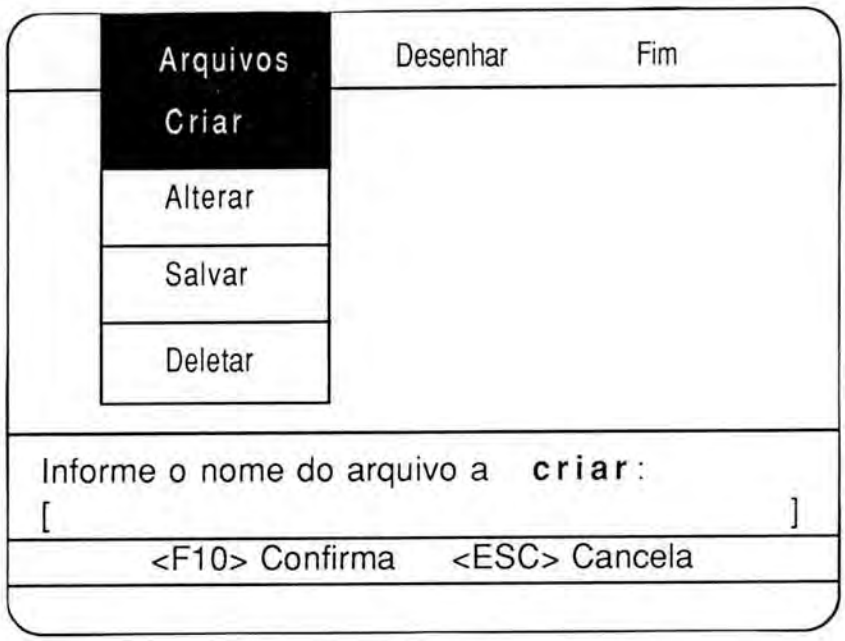

Figura 6.20 Seleção da opção ARQUIVOS

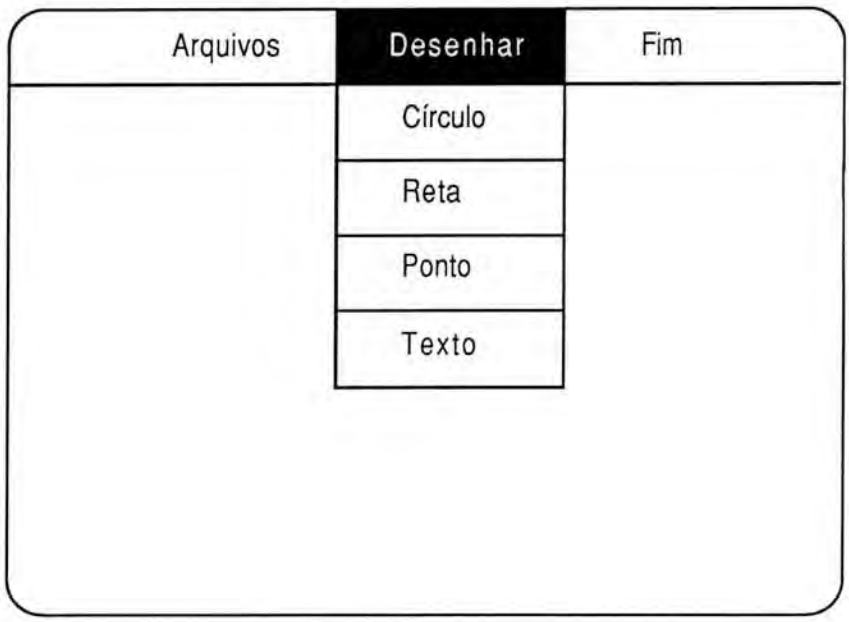

Figura 6.21 Seleção da opção DESENHAR

Figura 6.22 Seleção da opção CRIAR ARQUIVO

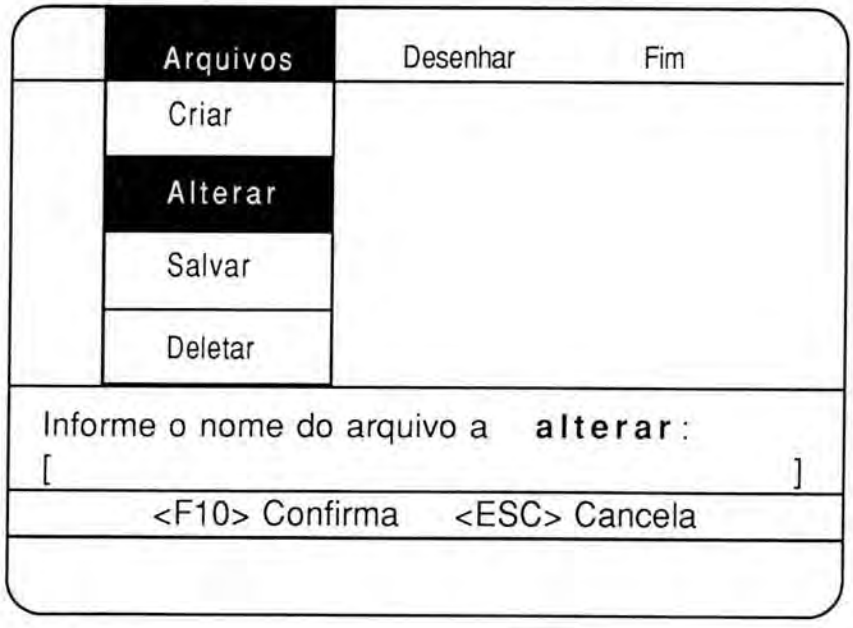

Figura 6.23 Seleção da opção ALTERAR ARQUIVO

Figura 6.24 Seleção da opção SALVAR ARQUIVO

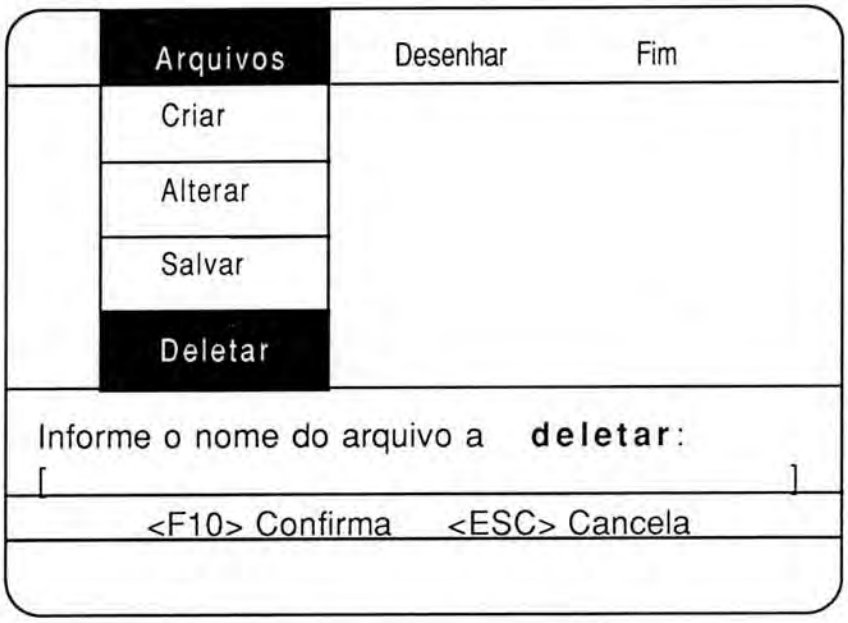

Figura 6.25 Seleção da opção DELETAR ARQUIVO

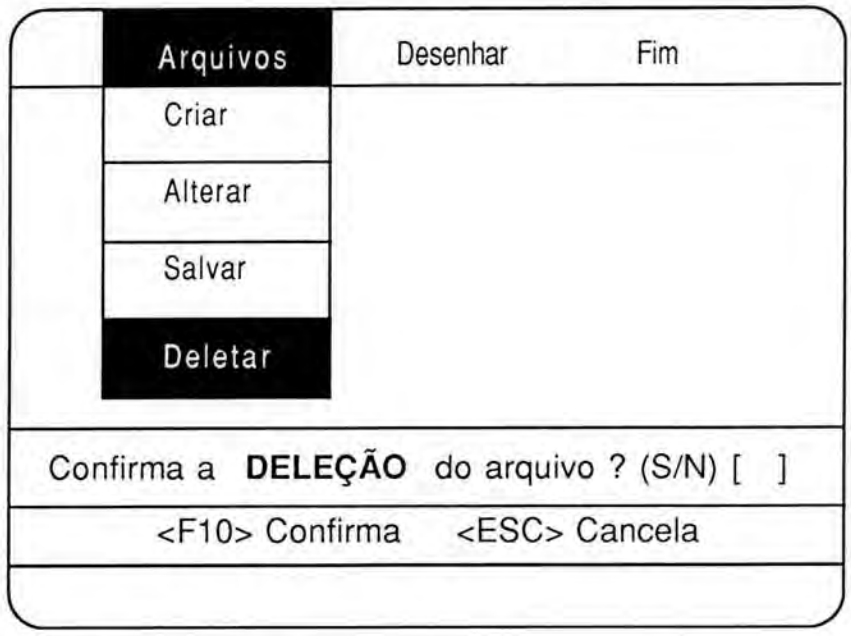

Figura 6.26 Seleção da opção confirmação da deleção do arquivo

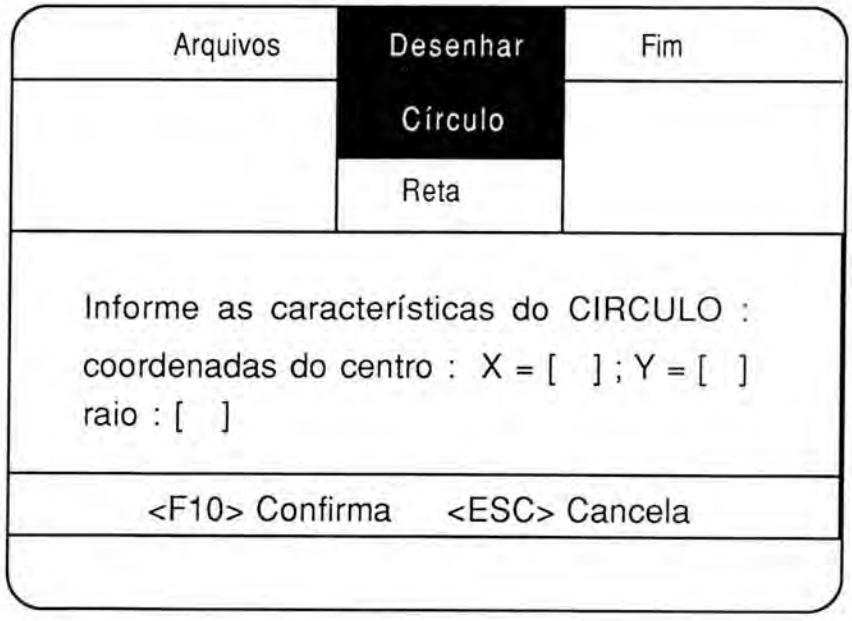

Figura 6.27 Seleção da opção CIRCULO DESENHAR

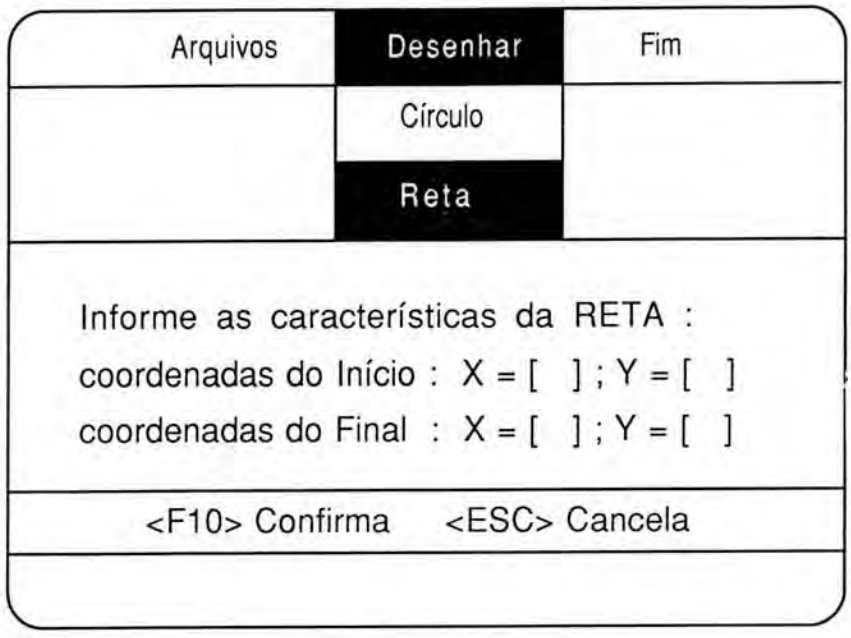

Figura 6.28 Seleção da opção RETA DESENHAR

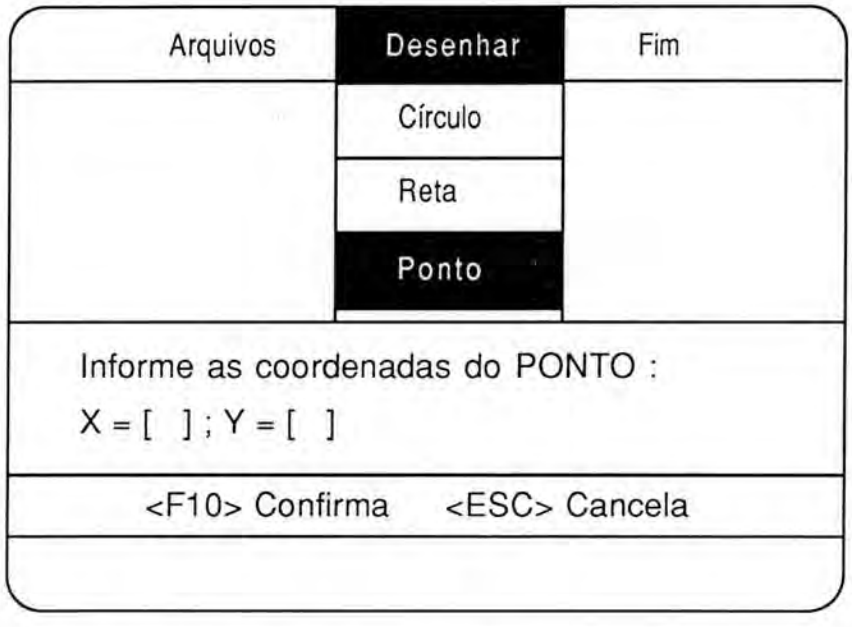

Figura 6.29 Seleção da opção PONTO DESENHAR

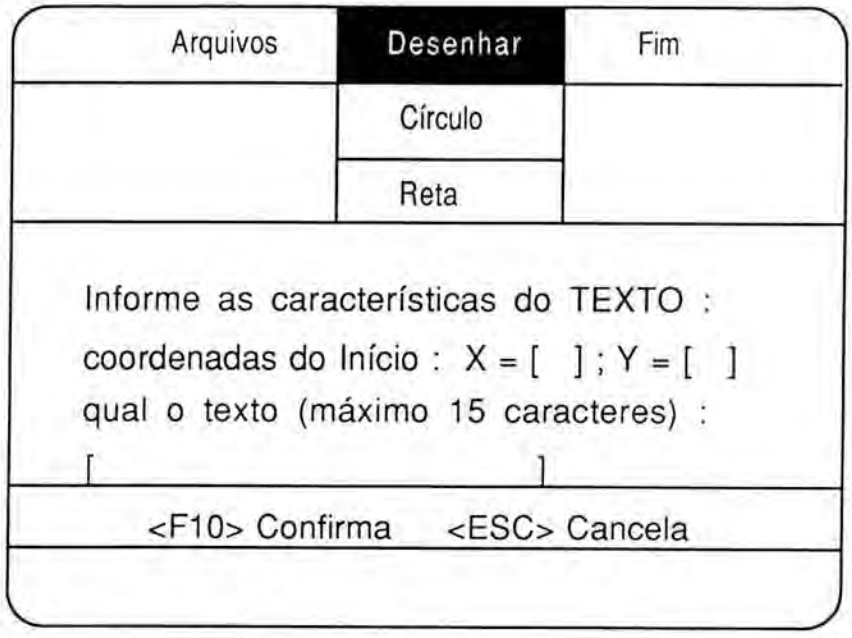

Figura 6.30 Seleção da opção TEXTO DESENHAR

6.14 Uso

A metodologia apresentada não é, nem propõe-se a ser, uma metodologia de projeto de sistemas. Sua área de atuação está restrita ao projeto da IU. Esta metodologia é utilizada pelo projetista físico do sistema, pois a determinação das características da IU só é possível após a realização do estudo lógico. e seja possivel a determinação das tarefas que o sistema executará. A partir deste momento a IU pode ser desenvolvida de forma paralela à programação das funções da aplicação, pois os formatos das chamadas das funções já foi estabelecida.

O principal problema desta proposta é a dificuldade de realizar-se uma validação por prototipagem. As únicas validações permitidas e previstas são as que o

usuário sugere quando apresentados os esboços dos elementos de interação (etapa 7).

No caso de serem necessárias alterações, a metodologia proposta apresenta o mesmo mecanismo de alterações oferecidos pelas metodologias de análise estruturada [GAN 83]. O ciclo de vida de uma IU segue o modelo usado para a análise de sistemas [RHY 86], assim sendo, modificações podem ser facilmente introduzidas, mas os efeitos são propagados para frente e para trás nas demais etapas do ciclo de vida. Para ilustrar, uma alteração no estilo de interação adotado num determinado ponto de interação (possível num SGIU), repercute em reavaliação das tarefas realizadas nas etapas de determinação dos estilos de interação, de esboços das apresentações, de descrição da IU e de tradução e compilação da descrição.

7 O PROTÓTIPO

Este capítulo descreve, em linhas gerais, o protótipo experimental que implementa a proposta GRAEDIUS. Este protótipo permite a descrição de IUs que se utilizem dos estilos de menus e formulários nas suas interações de entrada e figuras e textos simples nas suas interações de saida. O ambiente de implementação, as limitações impostas e encontradas e os requisitos mínimos detectados são descritas nas seções 7.1, 7.2 e 7.3, respectivamente. Uma rápida descrição das ferramentas utilizadas é feita na seção 7.4. As estruturas de dados utilizadas são apresentadas na seção 7.5. A seção 7.6 descreve a divisão modular utilizada na implementação do protótipo. Os procedimentos de uso e a interface do protótipo são discutidas na seção 7.7.

7.1 Ambiente de Implementação

Para a determinação do ambiente de implementação do protótipo, foi necessário a determinação das plataformas de hardware e software utilizados.

As plataformas de hardware disponíveis para a implementação do protótipo eram as baseadas em microcomputadores com o padrão IBM-PC [NOR 88] e as baseadas em estações de trabalho da SUN [SUN 89].

O ambiente de implementação adotado foi baseado na plataforma oferecida pelos microcomputadores com padrão IBM-PC, por possuirem uma base de software mais ampla e uma

maior difusão de uso. A base de software desta implementação é composta por ferramentas de gerenciamento de janelas, de menus, de formulários e um gerador de reconhecedores de GA.

7.2 Limitações

Como o protótipo é do tipo experimental, visando somente demonstrar que a ferramenta proposta é factível de uso, algumas limitações foram introduzidas. Estas limitações tiveram reflexos sobre os níveis de projeto e de implementação do protótipo.

As limitações introduzidas ao nível de projeto foram sobre os estilos de interação suportados, sendo estas limitadas à metáfora conversacional, ou seja, o protótipo suporta apenas alguns tipos de menus e formulários.

Algumas limitações foram impostas pelos gerenciadores dos estilos de menus e formulários adotados. Os tipos de menus suportados são os horizontais e os verticais de itens múltiplos (permitem a escolha de somente um item por vez). Nenhum estilo implementado suporta funções de ajuda.

No nível de implementação, limitações foram introduzidas na qualidade da apresentação da IU. A adoção do modo textual, oferecido pelo padrão IBM-PC, por ser este o modo manipulado pelos gerenciadores adotados. Porém esta escolha trouxe conseqüências na qualidade gráfica final

oferecida. Perdeu-se qualidade na apresentação das figuras geométricas utilizadas.

7.3 Requisitos

Para a implementação do protótipo tornou-se necessário o estabelecimento de requisitos mínimos para os níveis de hardware e software. Estes requisitos surgiram como conseqüência da determinação das necessidades e disponibilidades de software.

A determinação, localização e obtenção de ferramentas que auxiliassem na implementação do protótipo (ver 7.6) foram fatores decisivos na escolha da plataforma de hardware adotada. As ferramentas de software que se fizeram necessárias foram :

- um gerador de reconhecedores de GA;
- um gerenciador de janelas;
- um gerenciador genérico de menus;
- um gerenciador genérico de formulários;
- uma biblioteca com funções geométricas básicas; e
- um compilador da linguagem hospedeira utilizada.

Os gerenciadores de janelas, menus e formulários adotados estão escritos em linguagem C, e foram obtidos em forma de programas fonte escritos em Turbo C da Borland

Inc. [BOR 88a]. O gerador de reconhecedores de gramáticas localizado, permite também a geração de fontes escritos em Turbo C, portanto, o compilador adotado foi o compilador Turbo C.

Outro conjunto de ferramentas disponíveis para a plataforma adotada, é o fornecido pelo ambiente de programação WINDOWS 2.03[1] da Microsoft Inc. [WIN 88]. Seu uso foi abandonado devido a problemas encontrados na instalação de seu ambiente de apoio à programação. Estes problemas impediram a execução de testes preliminares. Além destes, indicações de pesquisadores desta instituição apontaram para outros que dificultariam a implementação deste protótipo. Estes fatores levaram a busca e adoção de gerenciadores "abertos" (com códigos fontes).

As características mínimas que o hardware deve possuir para permitir o uso das ferramentas de software selecionadas, foram determinadas basicamente pelas necessidades do compilador adotado. São elas :

- uma unidade de disco flexível;
- um disco rígido com 20 Mb;
- memória RAM de 640 Kb; e
- um monitor compatível com o padrão CGA.

---

[1] A versão 3.0 somente tornou-se acessível após o término da confecção do protótipo.

7.4 Ferramentas Utilizadas

A biblioteca de funções geométricas básicas tem um funcionamento bastante simples. Suas funções somente exibem nas janelas de saída indicadas os valores dos parâmetros recebidos. Os objetos "implementados" são o ponto, a reta, o retângulo, o círculo e texto. O anexo 5 apresenta a lista das funções disponíveis nesta biblioteca.

O gerenciador de janelas é composto por um conjunto de rotinas que realizam todas as operações necessárias sobre janelas. A sua capacidade de manipulação de janelas está limitada ao modo textual dos microcomputadores padrão IBM-PC. Uma listagem das funções disponíveis neste gerenciador é apresentada no anexo (?).

O gerenciador de menus adotado está baseado no gerenciador de janelas acima descrito. Originalmente oferecia funções que implementavam uma estrutura fixa de menus. Para que pudesse ser aproveitado neste protótipo, algumas funções tiveram de ser criadas e outras adaptadas, mas no final obteve-se um conjunto de funções que implementam menus horizontais e verticais que, dependendo do tipo de retorno associado a cada item, incluem uma cadeia de caracteres na memória do reconhecedor léxico da gramática de controle da IU, ou invocam a execução do elemento de interação associado. A descrição das funções oferecidas por este gerenciador encontra-se listada no anexo 3.

O gerenciador de formulários, de igual modo, está baseado no gerenciador de janelas descrito. Para possibilitar um melhor aproveitamento das potencialidades oferecidas e necessárias, muitas funções foram alteradas para que manipulassem as características dos diferentes

retornos associados às tarefas definidas para cada formulário. Uma listagem das funções oferecidas, dos parâmetros utilizados e das tarefas realizadas encontra-se no anexo 4.

Além das alterações já mencionadas, foram acrescidas funções que permitem a criação e inclusão de opções, tarefas e entradas de forma dinâmica nos menus e formulários manipulados.

O gerador de reconhecedores de GA, foi a ferramenta mais automatizada utilizada. Foi resultado da trabalho de conclusão da graduação de Fernando Raupp Rosa do curso de ciências da computação desta instituição de ensino. A ferramenta desenvolvida recebeu o nome de SinLex e uma melhor descrição de seu funcionamento e capacidades é encontrada em [ROS 89]. Esta foi a ferramenta que sofreu o menor número de alterações. Na essência, as alterações estão relacionadas com a inclusão de comandos nos reconhecedores gerados de forma a permitir que estes se adaptem melhor às caracteristicas de GRAEDIUS. A ferramenta assim alterada recebe neste trabalho o nome de SINGRAED.

7.5 Estrutura de Dados Utilizada

As estruturas de dados utilizadas na IU tem capacidade de representar as informações necessárias para a definição de :

- janelas; e

- elementos de interação, tais como :

  · menus;

  · formulários.

Para a definição das janelas é necessária a determinação de :

- nome de identificação;
- tamanho, em termos de linhas e colunas;
- o conjunto de cores válidas;

Para a definição de menus são necessários a determinação de :

- nome de identificação do menu;
- lista dos itens apresentados, composta por :

  · letra que permite a seleção direta da opção;

  · determinação do tipo do retorno associado;

  · o nome do elemento de interação que deve ser executado caso haja um encadeamento direto de elementos;

  · o texto de retorno que deve ser introduzido na memória do reconhecedor léxico do executor da gramática de controle da IU.

Para a definição de formulários são necessários a determinação de :

- nome de identificação do formulário;
- janela utilizada;
- lista dos comandos apresentados, composta por :
  - tecla que define o comando;
  - determinação do tipo do retorno associado;
  - o nome do elemento de interação que deve ser executado caso haja um encadeamento direto de elementos;
  - o texto de retorno que deve ser introduzido na memória do reconhecedor léxico do executor da gramática de controle da IU.
- lista dos campos solicitados, composta por :
  - tipo do campo;
  - tamanho do campo;
  - máscara de edição associada;
  - posição do campo dentro da janela utilizada;
  - valor "default";

A listagem destas estruturas são apresentadas no anexo 6.

Os diversos módulos que compõe o protótipo, possuem outras estruturas de dados. Estas são de utilização interna dos módulos e por serem irrelevantes na definição dos elementos manipulados pela IU não serão abordados neste documento. Se na IU forem necessárias estruturas de dados que permitam a manipulação de informações provenientes da aplicação, é de responsabilidade do projetista do sistema a sua definição, inclusão e determinação das operações válidas.

7.6 Estrutura Modular da Implementação

O protótipo é composto por dois módulos principais. A inter-relação existente entre eles é mostrada na figura 7.1.

Para facilitar a programação do protótipo foram definidos vários módulos. O critério adotado para a realização da divisão foi o tipo de tarefa realizada, o que levou a criação de módulos com tarefas e objetivos bem definidos. As sub-seções 7.6.1 à 7.6.4 apresentam as descrições de cada um destes módulos. Cada descrição apresenta, de forma suscinta, as ferramentas utilizadas, as tarefas envolvidas, os objetivos, as entradas e as saídas.

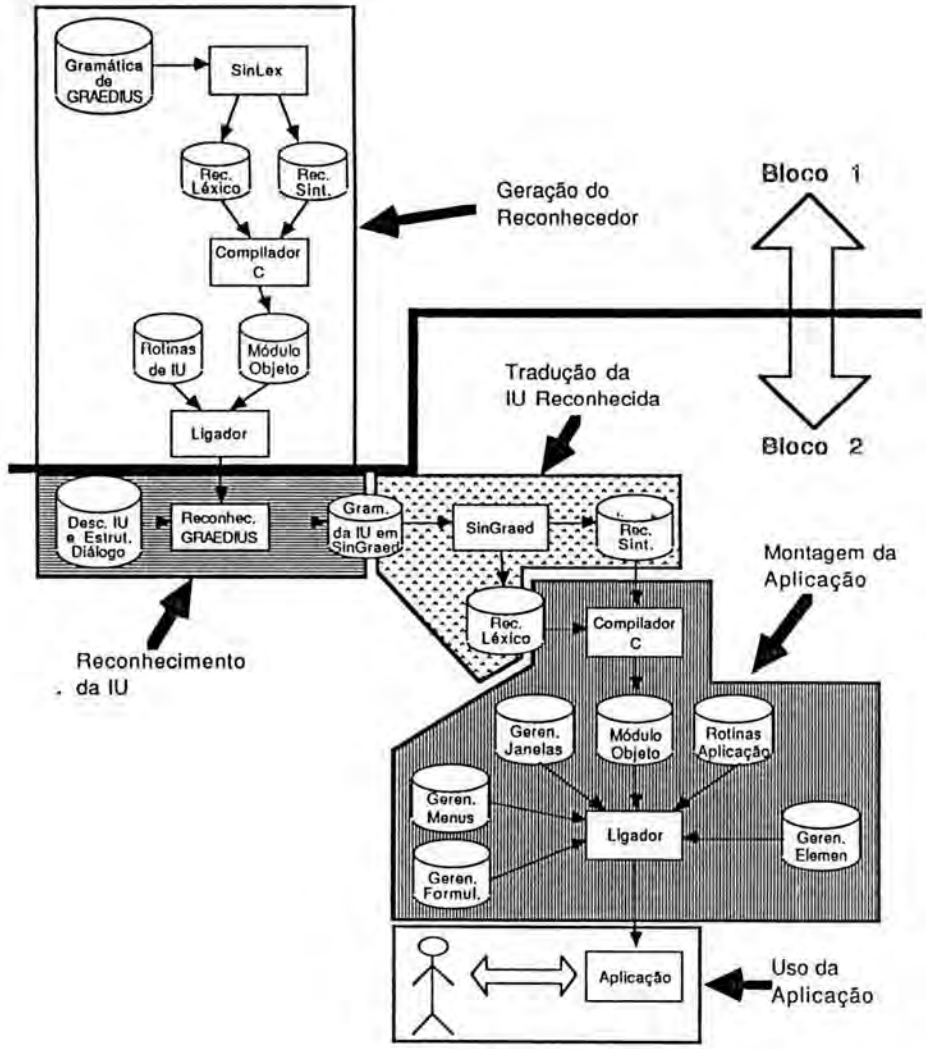

Figura 7.1 Inter-relação dos módulos do protótipo

O processo de desenvolvimento do protótipo apontou para a existência de dois momentos distintos, um de montagem do reconhecedor, e outro de montagem do sistema de aplicação. O primeiro momento é de responsabilidade dos meta-programadores de IUs, enquanto que o segundo momento é manipulado e desenvolvido pelos projetistas e programadores da aplicação. Uma interessante caracteristica do protótipo é a inexistência de um reconhecedor léxico explicito. As funções de reconhecimento léxico são supridas pelos gerenciadores de menus e formulários.

7.6.1 Geração do Reconhecedor

- Ferramentas Envolvidas

  - SinLex;

  - Compilador Turbo C.

- Tarefas Envolvidas

  - definir a gramática da linguagem GRAEDIUS (ver anexo 7).

  - executar o SinLex sobre a a gramática de definição da linguagem GRAEDIUS;

  - gerar os módulos de reconhecimento léxico e sintático da gramática de entrada;

  - compilar os módulos léxico e sintático gerados pelo SinLex com o compilador Turbo C;

  - ligar os módulos "objetos" obtidos com o módulo que define a IU para a geração do reconhecedor da linguagem GRAEDIUS.

- Objetivos

  - gerar um reconhecedor da linguagem de descrição de IU GRAEDIUS.

- Entradas

  - descrição da gramática da linguagem de descrição de IU GRAEDIUS.

- Saídas

  - programa executável de reconhecimento da linguagem de descrição de IU GRAEDIUS.

7.6.2 Reconhecimento da IU

- Ferramentas Envolvidas

  · Reconhecedor da linguagem GRAEDIUS;

- Tarefas Envolvidas

  · ler a gramática de definição da IU;

  · montar as estruturas de descrição da IU;

  · gerar uma gramática no formato SinLex que equivalha à IU descrita na entrada.

- Objetivos

  · gerar uma gramática, em formato SinLex, que descreva uma IU equivalente à descrita em GRAEDIUS.

- Entradas

  · descrição da IU na linguagem GRAEDIUS.

- Saídas

  · uma gramática no formato SinLex equivalente à IU descrita em GRAEDIUS.

7.6.3 Tradução da IU Reconhecida

- Ferramentas Envolvidas

  · SinGRAED;

- Tarefas Envolvidas

  · executar o SinGRAED sobre a gramática que define a IU descrita para o sistema, gerada pelo reconhecedor GRAEDIUS;

· gerar os módulos de reconhecimento léxico e sintático da gramática de entrada;

- Objetivos

· gerar um reconhecedor a partir da gramática de entrada que define a IU do sistema.

- Entradas

· uma gramática no formato SinLex equivalente à IU descrita em GRAEDIUS.

- Saídas

· códigos fontes dos módulos de reconhecimento léxico e sintático da IU descrita.

7.6.4 Montagem da IU

- Ferramentas Envolvidas

· Compilador Turbo C.

- Tarefas Envolvidas

· compilar os módulos léxico e sintático gerados pelo SinGRAED com o compilador Turbo C;

· ligar os módulos "objetos" obtidos com os gerenciadores usados pela GRAEDIUS para a geração do reconhecedor da linguagem GRAEDIUS;

· ligar o módulo do Gerenciador de Janelas à aplicação final;

· ligar o módulo do Gerenciador de Menus à aplicação final;

- ligar o módulo do Gerenciador de Formulários à aplicação final;
- ligar o módulo das Rotinas da Aplicação à aplicação final;
- ligar o módulo do Reconhecedor Sintático à aplicação final;

- Objetivos

- compor a aplicação final através da combinação das rotinas necessárias para implementar os estilos de interação, o controle do fluxo do diálogo e as rotinas da aplicação que implementam as tarefas oferecidas.

- Entradas

- módulo de rotinas para o gerenciamento de janelas;
- módulo de rotinas para o gerenciamento de formulários;
- módulo de rotinas para o gerenciamento de menus;
- módulo com as rotinas da aplicação;
- módulo do reconhecedor léxico da IU.
- módulo do reconhecedor sintático da IU.

- Saídas

- programa executável da aplicação.

7.7 Uso do Protótipo

A interface com o usuário utilizada pelo protótipo é bastante simples, sendo composta pela chamada do reconhecedor de GRAEDIUS, informando como parâmetros de entrada o nome do arquivo que contém o texto de definição da IU e, opcionalmente, o nome do arquivo que deve ser gerado como saida. A sintaxe de chamada é :

GRAEDIUS <nome_de_entrada> [<nome_de_saída>]

onde :

<nome_de_entrada> : é o nome do arquivo que contém a descrição da interface a ser reconhecida e gerada pelo sistema.

<nome_de_saida> : parâmetro opcional que especifica o nome do arquivo que conterá o código gerado para a interface reconhecida, se omitido é assumido o mesmo nome do arquivo de entrada.

8 CONCLUSÃO

Este capitulo apresenta na seção 8.1 uma discussão sobre as vantagens e desvantagens do uso de ferramentas SDIU, dando ênfase às inerentes a GRAEDIUS. A seção 8.2 enumera um conjunto de trabalhos futuros que visam completar a proposta GRAEDIUS, comparando-a a outras ferramentas encontradas na bibliografia. As conclusões sobre o trabalho desenvolvido são apresentadas na seção 8.3.

8.1 Vantagens e Desvantagens

Uma das principais vantagens procuradas com o desenvolvimento da GRAEDIUS foi o incremento da produtividade dos programadores de aplicação. Este incremento é resultado da grande facilidade experimentada na programação das IUs. Estima-se que uma conseqüência direta deste incremento de produtividade é uma diminuição drástica dos custos totais de desenvolvimento dos sistemas. Em sistemas de IA este custo está na ordem de 40-50% do total [MYE 89]. A grandeza da redução obtida pode ser facilmente determinada pela comparação dos custos e qualidades obtidas entre diferentes implementações de um sistema, uma que tenha utilizado a proposta GRAEDIUS e outra com programação convencional.

Apesar deste grande potencial, muitos construtores de SDIUs experimentais tem descoberto que seus sistemas não são muito fáceis de usar. O problema é ocasionado, parcialmente, pela complexidade inerente à própria ferramenta e também, parcialmente, motivado pela

pobre interface que oferecem [RHY 87]. O protótipo desenvolvido bem pode ser enquadrado nos SDIUs que apresentam problemas do segundo tipo. Uma forma de corrigir, ou pelo menos atenuar-se, este tipo de problema é alcançada pelo uso de sistemas de apoio ao desenvolvimento, sistemas que permitem a execução interativa da determinação da IU. Estes sistemas são, normalmente, conhecidos como *shells*, cascas, pois definem tarefas que permitem o uso de sistemas complexos com maior facilidade.

Uma grande vantagem na proposta é que possui todas as construções das linguagens de programação tradicionais. Isto possibilita seu uso para a definição de IUs mais complexas, pois as computações intermediárias necessárias podem ser facilmente resolvidas nas ações semânticas da gramática usada pelo componente de controle da IU. Alguns sistemas que necessitam destas caracteristicas são os sistemas de desenho assistido por computador (DAC), os de projeto de ambientes e os que usam comandos diretos [RHY 87].

A proposta GRAEDIUS define uma metodologia de projeto de IUs (ver 6) que lembra os princípios de construção de software nos seus aspectos mais abstratos. A grande inovação de GRAEDIUS é a introdução, no ambiente acadêmico da UFRGS, dos conceitos de SDIUs pela proposição de um sistema que permitirá uma melhor documentação, facilidade de especificação e manutenção das IUs dos sistemas desenvolvidos. Por ter sido implementada como expansão de GA, GRAEDIUS permite não somente a descrição da IU, mas também que seja dividida de forma que a aplicação seja responsável somente pela parte computacional da resolução das tarefas, enquanto que o controle da apresentação, da execução e as computações simples que

envolvem diretamente os elementos de interação ficam a cargo da GRAEDIUS.

De acordo com Myers [MYE 89], o uso de SDIUs traz algumas conseqüências diretas sobre duas áreas :

- resulta em melhores interfaces, trazendo como algumas vantagens imediatas :
  - a possibilidade de uma mesma aplicação possuir mais de uma interface, através do uso de mecanismos que permitam a associação em tempo de execução entre os pontos de interação e os elementos de interação disponiveis;
  - haver maior consistência entre várias aplicações, pois suas interfaces terão sido desenvolvidas com a mesma ferramenta, o que além da consistência diminui a necessidade de treinamento e aumenta a satisfação de uso para usuários esporádicos, pois transmite-lhes maior sensação de domínio dos sistemas;
  - maior facilidade de introdução de alterações na IU, decorrente da homogeneidade e formalismo de descrição das IUs utilizadas por vários sistemas;
  - facilidade no inter-relacionamento dos membros da equipe de projeto de interfaces, pois todos passam a adotar um vocabulário padrão.
- a criação é mais fácil e a manutenção das interfaces é mais econômica, tendo reflexos diretos sobre :

- o tempo e tipo de treinamento dos usuários necessário para o aprendizado de novos sistemas, pois como as IUs são homogêneas, quem conhece um sistema conhece a maioria (como no MACINTOSH);

- o código resultante será mais reutilizável, pois as IUs vão incorporar códigos comuns a várias rotinas que executa (gerenciamento dos elementos de interação, etc);

- as especificações das IUs poderão ser mais facilmente representadas, validadas e avaliadas, desde que sejam oferecidas ferramentas de apoio à representação, validação e avaliação;

- portabilidade será facilitada, pois as operações dependentes de dispositivo estarão concentradas na IU.

A proposta GRAEDIUS brinda o usuário com algumas facilidades que implicam na resolução ou correção das conseqüências apontadas por Myers. Pode-se dizer que no primeiro grupo de conseqüências, GRAEDIUS somente não oferece mecanismos de associação em tempo de execução dos pontos de interação com os elementos de interação.

Já nas conseqüências arroladas no segundo grupo, GRAEDIUS não oferece ferramentas para as tarefas de validação, avaliação e facilidades representação da IU, mas todos estas deficiências são apontadas como trabalhos futuros. Quanto à portabilidade, o uso de GRAEDIUS permite uma mais fácil portabilidade dos sistemas, pois somente o seu reconhecedor necessita ser alterado para que a IU esteja portada.

8.2 Trabalhos Futuros

A proposta deste trabalho representa um esforço inicial na construção de SDIUs genéricos na UFRGS, e portanto necessita do desenvolvimento de trabalhos complementares. Uma lista deste trabalhos é apresentada abaixo :

- **desenvolver um *shell* que facilite ao usuário do ambiente GRAEDIUS a definição, simulação e prototipação de IUs;** a finalidade seria a de desenvolver um ambiente que permitisse a definição interativa de IUs, usando GRAEDIUS como base formal e de armazenamento das especificações. Este tipo de facilidade permitiria uma maior e melhor interação do usuário final com os programadores da interface, o que acarretaria numa IU inicial mais próxima das necessidades do usuário.

- **adicionar a capacidades para a descrição e manipulação de elementos do estilo de interação de comando direto;** interação através de comando direto traz grandes vantagens sobre os outros estilos, principalmente quando a utilização do sistema for esporádica, ou o custo de treinamento associado elevado. Por ser de uso altamente intuitivo, este tipo de interação é muito solicitada. Quanto ao problema citado na bibliografia, [GRE 86] [MYE 89] [RHY 87], de que IUs que usem comando direto como estilo de interação não são facilmente descritas com gramáticas, devido às computações intermediárias, como Hudson e King já experimentaram, GAs podem ser usadas para descrever IUs deste tipo [HUD 86].

- **permitir que o usuário faça interativamente durante o uso alterações nos atributos físicos da apresentação da IU, afim de particularizar a IU aos seus gostos e preferências;** um bom exemplo para ilustrar as vantagens desta capacidade são os problemas de cores enfrentados em sistemas desenvolvidos em ambientes que possuem monitores multi-tonal (fósforo verde, p.ex.) e utilizados com monitores coloridos. Talvez este problema possa ser resolvido pelo emprego de tabelas que descrevam as características das IUs de cada usuário.

- **permitir ao usuário a capacidade de alterar, dinamicamente, os estilos de interação utilizados na IU, visando permitir um incremento na satisfação de uso e melhor adaptação à seu perfil;** esta capacidade está intimamente relacionada com os usuários sofisticados, pois quando iniciam o aprendizado de um sistema não se ressentem de utilizar estilos mais didáticos (menus, p.ex.), mas com o aumento de uso, estilos mais dinâmicos tornam-se necessários para aumentar a produtividade e satisfação do usuário. Novamente o uso de tabelas que mantenham as descrições das IUs particularizadas por usuário, parece ser a solução mais adequada.

- **criar um sistema de apoio para, em tempo de execução, dar apoio e conselhos ao usuário acerca de alterações que podem ser introduzidas na IU para lograr um aumento de sua produtividade, conforto de uso, etc;** o uso deste tipo de sistema serviria somente para orientar as alterações que o usuário deseja introduzir sobre a IU. Uma boa fonte de sugestões poderiam ser as estatísticas de uso e erros levantadas durante um período de uso.

- **criar uma definição intermediária que facilitasse a portabilidade das IUs criadas, não só para várias máquinas/ambientes, mas também para ser utilizada de forma semi-independente da linguagem de programação do sistema (algo como uma canônica intermediária);** linguagens canônicas são muito interessantes quando o ambiente em que será executada determinado sistema não for definivel em tempo de desenvolvimento do sistema, portanto é interessante que também a IU seja facilmente portável. Esta facilidade é obtida ou com o uso de linguagens canônicas [PIM 91], ou com a criação de um reconhecedor de GRAEDIUS para cada ambiente em que se deseja trabalhar.

8.3 Finalização

O sistema GRAEDIUS não se constitui numa proposta fechada e consistente, mas ao longo de sua elaboração buscou-se sempre uma generalização que permitisse sua expansão e adaptação. Os trabalhos propostos foram apresentados de forma ordenada por prioridade de execução, pois com sua confecção obter-se-á uma ferramenta GRAEDIUS ao mesmo nivel das citadas na bibliografia (SYNGRAPH, TIGER, SASSAFRAS e COUSIN entre outros) [MYE 89]. As implementações destas sugestões estão longe de serem impensáveis ou muito complexos, são antes algo laboriosos.

Como apresentado no capítulo 0, IUs podem ser estudadas como linguagens que permitem ao usuário definir as tarefas que deseja que o sistema execute e ao sistema apresentar os resultados das execuções destas tarefas. Esta interpretação de IUs levou o desenvolvimento da proposta GRAEDIUS a partir de GAs, pois este tipo de gramática além de permitir a definição dos componentes léxicos e sintáticos das linguagens que descrevem, também descrevem a semântica estática das linguagens. Esta capacidade, permite

que GRAEDIUS defina as computações simples que estejam associadas aos elementos de interação.

O protótipo implementado demonstrou a viabilidade da metodologia apresentada no capítulo 6, assim como indicou a necessidade de alguns trabalhos a serem desenvolvidos no futuro. O reduzido número de estilos oferecidos pelo protótipo, não permitiu que fossem testadas todas as capacidades de GRAEDIUS, mas foram suficientes para demonstrar que o seu uso é viável, apesar de complexo. As principais deficiências levantadas estão indicadas na seção 8.2. Outros dois grandes problemas levantados durante a confecção do protótipo foram a falta de uma ferramenta que facilitasse a definição dos elementos de interação e a inexistência de mecanismos de validação da proposta. A validação é, então, realizada através de uso exaustivo do protótipo, ou ambiente que a implemente.

ANEXO 1

Um resumo das etapas que devem ser executadas para o projeto de IUs.

Tabela : etapas X tarefas

1-determinação do perfil médio dos usuários.	O objetivo deste perfil é o de possibilitar uma melhor adequação da estrutura de diálogo e dos estilos de interação, postos a disposição, às reais necessidades do usuário.
2-determinar as tarefas executadas pelo sistema.	A determinação das tarefas auxilia no dimensionamento do sistema e no projeto da estrutura do diálogo.
3-determinar a estrutura do diálogo.	A estrutura do diálogo adotada determina qual a metáfora de diálogo que será adotada e o conjunto de estilos de interação que poderão ser utilizados pela IU.
4-especificar os pontos de interação.	Os pontos de interação são definidos sobre a estrutura do diálogo de modo a definir os momentos em que o usuário terá que intervir na execução do sistema.
5-determinar quais os estilos de interação que melhor se adaptam à comunidade de usuários da aplicação.	Com base no perfil médio da comunidade de usuários, é mais fácil determinar quais os estilos de interação que melhor se adequarão às suas necessidades.

6-definir os parâmetros de cada tarefa.	A definição dos parâmetros facilita a correta definição das apresentações dos estilos de interação necessários em cada um dos pontos de interação marcados.
7-fazer um esboço da apresentação do estilo de interação usado em cada ponto de interação.	Estes esboços são feitos com base no perfil levantado e nas tarefas e seus parâmetros, visando auxiliar a descrição da apresentação da IU.
8-especificar as ligações entre a IU e a aplicação.	As ligações são feitas sobre a estrutura do diálogo, indicando quais e quando devem ser computadas as operações que executam, corretamente, as tarefas disponíveis no sistema.
9-descrever a IU em linguagem GRAEDIUS.	Esta descrição é feita em um nível de abstração que permite ao projetista da IU, não se preocupar com os detalhes da implementação. A descrição vai servir de base para a geração do módulo de gerenciamento da IU do sistema.
10-compilar a descrição da IU em GRAEDIUS.	A proposta GRAEDIUS é um SDIU, e portanto seu reconhecedor traduz as descrições das IUs em módulos de linguagem hospedeira.
11-compilar os módulos gerados.	Como GRAEDIUS gera módulos escritos em uma linguagem hospedeira, estes devem ser compilados para que com sua execução obtenha-se o sistema desejado.

ANEXO 2

Resumo das declarações das funções oferecidas pelo gerenciador de janelas. A cada declaração segue uma explanação de seus objetivos, parâmetros de entrada e saída.

Nome da Função :

CRIA_JANELA

Sintaxe da Função :

```
WINDOW *cria_janela(int  x,
                    int  y,
                    int  alt,
                    int  larg,
                    int  backgr,
                    int  foregr,
                    char *titulo)
```

Objetivo :

A função acima inicializa uma estrutura de janela com os parâmetros referenciados acima. Na inicialização é feito a alocação da área de salvamento e a inclusão na lista de janelas.

Parâmetro de Entrada :

```
x      : Posição horizontal da janela;
y      : Posição vertical da janela;
h      : Altura da janela;
w      : Largura da janela.
alt    : Altura da janela;
larg   : Largura da janela
backgr : Cor de fundo da janela
foregr : Cor do texto da janela
titulo : Titulo da janela
```

Parâmetro de Saída :

Ponteiro para a estrutura da janela criada

ANEXO 3

Resumo das declarações das funções oferecidas pelo gerenciador de menus. A cada declaração segue uma explanação de seus objetivos, parâmetros de entrada e saída.

Nome da Função :

CRIA_MENU

Sintaxe da Função :

```
GRMENU *cria_menu (char   *menu,
                   int    posx,
                   int    posy,
                   char   *cor,
                   char   *help,
                   int    tipo,
                   int    op_default,
                   WINDOW *wnd)
```

Objetivo :

Cria uma estrutura de descrição do menu, de acordo com as informações passadas pelos parâmetros de entrada.

Parâmetros de Entrada:

menu	: Nome do menu;
posx	: Posição horizontal do menu;
posy	: Posição vertical do menu;
cor	: Nome do conjunto de cores utilizáveis;
help	: Nome do rótulo do texto de ajuda associado;
tipo	: Tipo do menu = Vertical ("v") ou Horizontal ("h");
op_default	: Número da opção default;
wnd	: Ponteiro para a janela que conterá o menu;

Parâmetros de Saída:

Ponteiro para a estrutura de descrição de menu criada

Nome da Função :

CRIA_OPCAO

Sintaxe da Função :

```
void cria_opcao (GRMENU *menu,
                 char   *opcao,
                 int    id,
                 int    n_ordem,
                 int    hab,
                 char   *help,
                 int    tipo_ret,
                 void   (*funcao) (),
                 char   *token,
                 GRMENU *mn)
```

Objetivo :

Cria uma estrutura de descrição da opção, para um menu, de acordo com as informações passadas pelos parâmetros de entrada.

Parâmetros de Entrada :

menu	: Ponteiro para a estrutura de descrição de menu;
opcao	: Nome externo da opção;
id	: Número do caractere de identificação da opção;
n_ordem	: Número de ordem da opção no menu;
hab	: indicativo de habilitação/desabilitação.
help	: Nome do rótulo do texto de ajuda;
tipo_ret	: Indicativo do tipo de retorno associado a opção;
funcao	: Função que deve ser executada como retorno;
token	: Token que deve ser criado como retorno;
mn	: Menu que deve ser executado como retorno;

Parâmetros de Saída :

Ponteiro para a estrutura de descrição de menu criada.

Nome da Função :

DELETA_MENU

Sintaxe da Função :

```
void deleta_menu (GRMENU *menu)
```

Objetivo :

Elimina uma estrutura de descrição de menu.

Parâmetros de Entrada :

menu : Ponteiro para o menu;

Parâmetros de Saída :

Nenhum.

Nome da Função :

SETA_COR

Sintaxe da Função :

```
void seta_cor(WINDOW *janela,
              char   *cor)
```

Objetivo :

Com o nome do conjunto de cores validas, fixa as cores correspondentes a cada elemento.

Parâmetros de Entrada :

janela : Ponteiro para a estrutura de descrição da janela;
cor : Nome do conjunto de cores a ser utilizado;

Parâmetros de Saída :

Nenhum.

Nome da Função :

DESENHA_MENU

Sintaxe da Função :

```
void desenha_menu (GRMENU *menu,
                   int    *interv,
                   int    sel_default)
```

Objetivo :

Desenha na tela o menu indicado.

Parâmetros de Entrada :

menu : Ponteiro para a estrutura de descrição de menu;
interv : Tamanho do intervalo entre inicios das opções;
sel_default: Número da opção default;

Parâmetros de Saida :

Nenhum.

Nome da Função :

MOSTRA_MENU_HORIZONTAL

Sintaxe da Função :

```
int mostra_menu_horizontal (GRMENU *menu,
                            int    *hsel,
                            int    *n_buffer)
```

Objetivo :

Gerenciar a exibição de menus horizontais.

Parâmetros de Entrada :

menu : Ponteiro para a estrutura de descrição de menu;
hsel : Ponteiro para o número da opção selecionada;

n_buffer : Ponteiro para o número de tokens no buffer léxico;

Parâmetros de Saída :

Número da opção selecionada.

Nome da Função :

MOSTRA_MENU_VERTICAL

Sintaxe da Função :

```
int mostra_menu_vertical (GRMENU *menu,
                          int    *hsel,
                          int    *n_buffer)
```

Objetivo :

Gerenciar a exibição de menus verticais.

Parâmetros de Entrada :

menu : Ponteiro para a estrutura de descrição de menu;
hsel : Ponteiro para o número da opção selecionada;
n_buffer : Ponteiro para o número de tokens no buffer léxico;

Parâmetros de Saída :

Número da opção selecionada.

ANEXO 4

Resumo das declarações das funções oferecidas pelo gerenciador de formulários. A cada declaração segue uma explanação de seus objetivos, parâmetros de entrada e saída.

Nome da Função :

CRIA_FORM

Sintaxe da Função :

```
GRFORM *cria_form (char   *form,
                   int    posx,
                   int    posy,
                   char   *cor,
                   char   *help,
                   int    op_default,
                   WINDOW *wnd)
```

Objetivo :

Cria uma estrutura de descrição do formulário, de acordo com as informações passadas pelos parâmetros de entrada.

Parâmetro de Entrada :

form	: Nome do formulário;
posx	: Posição horizontal do formulário;
posy	: Posição vertical do formulário;
cor	: Nome do conjunto de cores validas;
help	: Nome do rótulo de ajuda associado;
op_default	: Número da tarefa default do formulário;
wnd	: Ponteiro para a janela que contém o formulário;

Parâmetro de Saida :

Ponteiro para a estrutura de descrição do formulário criado.

Nome da Função :

CRIA_TAREFA

Sintaxe da Função :

```
void cria_tarefa (GRFORM *form,
                  int    posx,
                  int    posy,
                  char   *nome,
                  int    tecla,
                  int    hab,
                  int    tipo_ret,
                  void   (*funcao) (),
                  char   *token)
```

Objetivo :

Cria uma estrutura de descrição da tarefa, para um formulário, de acordo com as informações passadas pelos parâmetros de entrada.

Parâmetro de Entrada :

form	: Ponteiro para a estrutura de descrição de formulário
posx	: Posicao horizontal do titulo da tarefa no formulário
posy	: Posição vertical do título da tarefa no formulário
nome	: Texto do titulo associado
tecla	: Tecla de seleção da tarefa
hab	: Indicativo do habilitação/desabilitação da tarefa
tipo_ret	: Indicativo do tipo de retorno
funcao	: Função que deve ser executada como retorno
token	: Token que deve ser criado como retorno

Parâmetro de Saída :

Nenhum.

Nome da Função :

CRIA_ENTRADA

Sintaxe da Função :

```
void cria_entrada (GRFORM *form,
                   int    posx,
                   int    posy,
                   char   tipo_ent,
                   char   *nome,
                   char   tipo_campo,
                   char   tam_cpo,
                   char   *mascara,
                   char   prec_cpo,
                   char   *v_default,
                   char   *help,
                   int    hab)
```

Objetivo :

Cria uma estrutura de descrição da entrada, para um formulário, de acordo com as informações passadas pelos parâmetros de entrada.

Parâmetro de Entrada :

form	: Ponteiro para a estrutura de descrição de formulário;
posx	: Posição horizontal do título da tarefa no formulário;
posy	: Posição vertical do título da tarefa no formulário;
tipo_ent	: Indicativo do tipo de entrada : rótulo/campo de dados;
nome	: Texto do rótulo associado;
tipo_campo	: Tipo de campo de dados : int/float/cadeia/data/bool;
tam_cpo	: Tamanho do campo de dados;
mascara	: Ponteiro para a máscara de edição do campo de dados;
prec_cpo	: Número de digitos na parte fracionária do campo float;
v_default	: Ponteiro para o valor default do campo de dados;
help	: Ponteiro para o rótulo do texto de ajuda associado;
hab	: Indicativo de habilitação/desabilitação da entrada;

Parâmetro de Saida :

Nenhum.

Nome da Função :

DELETA_FORM

Sintaxe da Função :

void deleta_form (GRFORM *form)

Objetivo :

Retira o formulário da pilha de elementos, eliminando todos os elementos de geração que estão "acima" dele e elimina a estrutura que descreve o formulário informado.

Parâmetro de Entrada :

form : Ponteiro para o formulário;

Parâmetro de Saida :

Nenhum.

Nome da Função :

DESENHA_FORM

Sintaxe da Função :

void desenha_form(GRFORM *form)

Objetivo :

Desenha o formulário especificado na tela.

Parâmetro de Entrada :

menu : Nome do menu;

Parâmetro de Saida :

Nenhum.

Nome da Função :

MOSTRA_FORM

Sintaxe da Função :

```
int mostra_form (GRFORM *form,
                 int    *trf_esc,
                 int    *n_buffer)
```

Objetivo :

Gerencia o recebimento das informações para o formulário informado.

Parâmetro de Entrada :

form	: Ponteiro para o formulário
trf_esc	: Ponteiro para o número da tarefa selecionada
n_buffer	: Número de tokens no buffer léxico

Parâmetro de Saída :

Indicação de escolha ou abandono.

ANEXO 5

Resumo das declarações das funções oferecidas pelo gerenciador de elementos de interação. A cada declaração segue uma explanação de seus objetivos, parâmetros de entrada e saída.

Nome da Função :

INICIALIZA_GER_ELEM

Sintaxe da Função :

```
void inicializa_ger_elem()
```

Objetivo :

Inicialização das variáveis locais.

Parâmetros de Entrada :

Nenhum.

Parâmetros de Saída :

Nenhum.

Nome da Função :

EXISTE_NA_PILHA

Sintaxe da Função :

```
int existe_na_pilha(char   tipo,
                    GRMENU *menu,
                    GRFORM *form)
```

Objetivo :

Partir do topo da pilha procura a ocorrência do elemento informado como parâmetro. Se encontra retorna TRUE, senão retorna FALSE.

Parâmetros de Entrada :

tipo	: Tipo de elemento a verificar : menu ou formulário;
menu	: Ponteiro para a estrutura de descrição de menu;
form	: Ponteiro para a estrutura de descrição de formulário;

Parâmetros de Saída :

Booleano que indica a existência ou não do menu.

Nome da Função :

MOSTRA_ELEM

Sintaxe da Função :

```
int mostra_elem (char   tipo_elem,
                 GRMENU *menu,
                 GRFORM *form,
                 int    *escolha,
                 int    *n_buffer)
```

Objetivo :

Inclue o elemento na pilha de elementos exibidos, se já está na pilha, elimina todos os elementos acima dele, re-exibe todos os elementos da pilha a partir da base e chama a execução da rotina correspondente ao elemento a ser mostrado.

Parâmetros de Entrada :

tipo_elem	: Tipo de elemento a verificar : menu ou formulário;
menu	: Ponteiro para a estrutura de descrição de menu;

form : Ponteiro para a estrutura de descrição de formulário;
escolha : Ponteiro para a tarefa/opção ativa;
n_buffer : Ponteiro para o número de tokens no buffer léxico;

Parâmetros de Saída :

Número da opção selecionada.

Nome da Função :

ESCONDE_ELEM

Sintaxe da Função :

```
void esconde_elem (char   tipo_elem,
                   GRMENU *menu,
                   GRFORM *form)
```

Objetivo :

Se o elemento indicado se encontra na pilha, elimina-o e todos os que estão "acima" dele, retirando-os da pilha.

Parâmetros de Entrada :

tipo_elem : Tipo de elemento a verificar : menu ou formulário
menu : Ponteiro para a estrutura de descrição de menu
form : Ponteiro para a estrutura de descrição de formulário

Parâmetros de Saída :

Nenhum.

ANEXO 6

Estrutura de dados utilizada no protótipo de GRAEDIUS.

```
/* Defines */

#ifndef ON
#define ON              1
#endif

#ifndef OFF
#define OFF             0
#endif

#define TAM_TOKEN       20      /* tamanho maximo de um token */

#ifndef FALSE
#define FALSE           0
#endif

#ifndef TRUE
#define TRUE            ~FALSE  /* TRUE eh a negacao de FALSE */
#endif

#define TIPO_MENU       0
#define TIPO_FORM       1

#define RETORNO_TOKEN   0
#define RETORNO_FUNCAO  1
#define RETORNO_ELEM    2
#define MENU_VERTICAL   3
#define MENU_HORIZONTAL 4

#define ROTULO          0
#define CAMPO           1
#define INTEIRO         2
#define CADEIA          3
#define FLOAT           4
#define DATA            5
#define BOOL            6

/* Definicoes das estruturas utilizadas pelo reconhecedor de GRAEDIUS */

typedef struct {
 char nome[30]; /* nome do conjunto de cores      */
 int  letra;    /* cor da letra                   */
 int  fundo;    /* cor do fundo                   */
 int  borda;    /* cor da borda                   */
```

```
45    int  sombra;    /* cor da sombra                  */
      int  car_id;    /* cor do carater de identificacao */
      int  off;       /* cor do desabilitado            */
     } TAB_CORES;

50   typedef struct {
      char   nome[30];/* nome da janela                                  */
      WINDOW *wnd;    /* ponteiro para a estrutura que descreve a janela */
      int    tam_x;   /* tamanho horizontal da janela                    */
      int    tam_y;   /* tamanho vertical da janela                      */
55    char   cor[30]; /* nome do conjunto de cores validas para a janela */
      int    pos_x;   /* posicao horizontal da janela                    */
      int    pos_y;   /* posicao vertical da janela                      */
     } TAB_JAN;

60   typedef struct opm{
      char          msg [80]; /* nome da opcao                                  */
      char          id;       /* num. car. de identificacao da opcao            */
      char          n_ordem;  /* classificacao da opcao no menu                 */
      int           hab;      /* indicativo de habilitacao da opcao             */
65    int           tipo_ret; /* tipo de retorno : funcao, token ou elemento */
      void          (*funcao_ret) (); /* funcao/elemento a executadar           */
      char          token[TAM_TOKEN]; /* token que deve ser gerado              */
      char          help[30]; /* rotulo do texto de ajuda associado             */
     } OPCOES;
70
     typedef struct grmenu{
      char   mname[80];   /* nome do menu                                        */
      char   posx,        /* coordenada x do canto superior esquerdo do menu */
             posy;        /* coordenada y do canto superior esquerdo do menu */
75    char   cor[30];     /* nome do conjunto de cores a serem utilizadas        */
      char   help[30];    /* rotulo do texto de ajuda associado                  */
      int    tipo;        /* tipo : VERTICAL ou HORIZONTAL                       */
      WINDOW *janela;     /* ponteiro para a janela a que pertence               */
      int    nro_opcoes;  /* numero de opcoes associadas ao menu                 */
80    int    op_default;  /* numero da opcao default                             */
      OPCOES *opcoes;     /* estrutura de opcoes                                 */
     } GRMENU;

     typedef struct {
85    int    posx,        /* posicao horizontal do rotulo na janela      */
             posy;        /* posicao vertical do rotulo na janela        */
      char   *rotulo;     /* ponteiro para o texto que compoe o rotulo   */
      int    tecla;       /* valor da tecla associada `a tarefa          */
      char   hab;         /* indicativo de habilitacao/desabilitacao     */
90    char   tipo;        /* tipo de retorno : token, funcao ou elemento */
      char   *token;      /* ponteiro para o token que deve ser retornado */
      void   (*func_ret)();/* ponteiro para a funcao/elemento a executar  */
     } TAREFA_FF;

95   typedef struct {
      int    posx,        /* posicao horizontal do rotulo na janela      */
             posy;        /* posicao vertical do rotulo na janela        */
      char   tipo_entrada; /* tipo da entrada : rotulo ou campo de dados */
```

```
        char   *rotulo;      /* ponteiro para o texto que compoe o rotulo      */
100     char   hab;          /* indicativo de habilitacao/desabilitacao        */
        char   tipo_cpo;     /* tipo do campo de dados : int/float/data/str/bool*/
        char   tam;          /* tamanho do campo de dados                      */
        char   prec;         /* numero de casas na parte decimal do float      */
        char   *campo;       /* ptr para o valor default do campo de dados     */
105     char   *mascara;     /* ptr para o mascara do campo de dados           */
        char   *help;        /* ptr para o rotulo do texto de ajuda            */
       } ENTRADA_FF;

       typedef struct {
110     char       *nome;     /* ponteiro para o nome do formulario            */
        int        posx,      /* posicao horizontal do formulario              */
                   posy;      /* posicao vertical do formulario                */
        char       *cor;      /* ptr para o nome do cjto. de cores utilizaveis */
        WINDOW     *janela;   /* ptr para a janela a que pertence              */
115     char       *help;     /* ptr para o rotulo do texto de ajuda           */
        int        op_default;/* numero da opcao default                       */
        int        n_trf;     /* numero de tarefas validas no formulario       */
        TAREFA_FF  *tarefas;  /* ponteiro para o vetor de tarefas do formulario */
        int        n_ent;     /* numero de entradas necessarias no formulario  */
120     ENTRADA_FF *entradas; /* ponteiro para o vetor de entradas do formulario*/
       } GRFORM;
```

ANEXO 7

Listagem da gramática de definição da linguagem GRAEDIUS implementada pelo protótipo.

```
SEC_LEXICA

/*************************************************************/
/*     Definicao dos conjuntos da gramatica lexica           */
/*************************************************************/

  CONJUNTOS

     Letra     -> "A".."Z", "a".."z", "_".
     Digito    -> "0".."9".
     AlfaNum   -> Letra, Digito.
     ConjCom1  -> #01..#41,#43..#255,EOLN.
     ConjCom2  -> #01..#41,#43..#46,#48..#255,EOLN.
     ConjPar   -> #32..#40,#42..#255.
     Coment1   -> " ".."z" , "|", "{" , EOLN.
     Coment2   -> " ".".", "0".."}" , EOLN.
     Aspas     -> #34.
     ConjCda   -> " ".."!", "#"..."}".
     ConjStr   -> " ".."&", "(".."}".
     IGNORE    -> EOLN," ",#09.

/*************************************************************/
/*      Definicao das producoes da gramatica lexica          */
/*************************************************************/

  REGRAS_LEXICAS

     INTEIRO     -> Digito INTEIRO       | Digito.

     FLOAT       -> Digito Float1        | "." Float2.
     Float1      -> Digito Float1        | "." Float2.
     Float2      -> Digito Float2        | Digito.

     CARACTER    -> "#" Carac.
     Carac       -> Digito Carac         | Digito.

     IDENT       -> Letra Ident_         | Letra.
     Ident_      -> AlfaNum Ident_       | AlfaNum.

     ENDER_VAR   -> "&" Ender_V1.
     Ender_V1    -> Letra Ender_V2       | Letra.
     Ender_V2    -> AlfaNum Ender_V2     | AlfaNum.

     STRING      -> "'" CorpoString.
```

```
45      CorpoString -> ConjStr CorpoString  | "'" StrGuampa
                     |  "'".
        StrGuampa   -> "'" CorpoString.

        CADEIA      -> Aspas CorpoCadeia.
50      CorpoCadeia -> ConjCda CorpoCadeia  | Aspas Str_Aspas
                     |  Aspas.
        Str_Aspas   -> Aspas CorpoCadeia.

        COMENTARIO  -> "/" Asterisco.
55      Asterisco   -> "*" CorpoComent.
        CorpoComent -> ConjCom1 CorpoComent | "*" TestaFim.
        TestaFim    -> "*" TestaFim         | ConjCom2
                                              CorpoComent
                     |  "/".
60
        AeS         -> "A" AeS1.
        AeS1        -> "&" AeS2.
        AeS2        -> "S".

65      ACAO_SEMAN  -> "{" AcSem1.
        AcSem1      -> Coment1 AcSem1       | "}".

    SEC_TOKENS

70    DELIMITADORES
        ";", ",", ".","..", "...", ":", "=", "->","+",
        "-", "*", "/", "&",   "|", "[", "]", "(", ")" .

      TOKENS
75      INTEIRO.
        FLOAT.
        CARACTER.
        ACAO_SEMAN.
        STRING.
80      CADEIA.
        AeS.
        VAL_VAR.
        ENDER_VAR.
        IDENT           EXCETO
85
     /*2*/    "Def_Interface","Fim_Def",
     /*5*/         "lexicos",
                      "conjuntos",
                      "automatos",
90                    "tokens",
                         "exceto",
     /*1*/    "fim",
     /*1*/    "apresentacao",
     /*1*/         "cores",
95   /*4*/         "janela",
                      "area",
                      "cor",
                 "mascaras",
     /*2*/    "geracao","ajuda",
100  /*1*/         "entrada",
    /*10*/             "menu",
```

```
                        "local",
                        "direcao",
                        "h","horizontal",
105                     "v","vertical",
                        "posicao",
                        "ativa",
                        "opcoes",
    /*11*/            "formulario",
110                     "comandos",
                        "int",
                        "float",
                        "char",
                        "data",
115                     "bool",
                            /*"v",*/"verdadeiro",
                            "f","falso",
     /*4*/            "ling_comando",
                        "prompt",
120                     "digitar","em",
     /*1*/        "saida",
     /*8*/    "sintaxe","abstrata","semantica",
                "global","acoes_in","acoes_out","void",
                "double",
125  /*3*/    "regras","de","associacao",
    /*12*/    "assinaturas",
              "const","signed","unsigned","static",
              "extern","register","short","long","far",
              "near","huge".
130 /*--*/
    /*66 palavras reservadas.*/

    SEC_SINTATICA

135 /************************************************/
    /*                                              */
    /*    Edson Gellert Schubert - 01/10/90 - CPGCC */
    /*    matr.: 013/88                             */
    /*                                              */
140 /************************************************/

    GRAEDIUS        -> ID_GRAEDIUS
                       LEXICOS
                       ASSINATURAS
145                    APRESENTA
                       GERACAO
                       REGRAS_ASSOC
                       ABST_SEMAN
                       FIM_GRAEDIUS
150                    EOF .
    ID_GRAEDIUS     -> "Def_Interface" ":"
                       IDENT
                       ";" .
    FIM_GRAEDIUS    -> "Fim_Def" "." .
155
    LEXICOS         -> "lexicos"
                       LEX_CONJUNTO
                       LEX_AUTOMATO
```

```
                        LEX_TOKEN
160                     FIM_SECCAO
                    |   .

    LEX_CONJUNTO    -> "conjuntos"
                        LST_CONJUNTO
165                 |   .
    LST_CONJUNTO    -> IDENT
                        "->"
                        CORPO_CJTO
                        ";"
170                     LST_CONJUNTO
                    |   .
    CORPO_CJTO      -> CADEIA
                        OUTRA_CADEIA
                        CORPO_CJTO_
175                 |   CARACTER
                        OUTRO_CAR
                        CORPO_CJTO_
                    |   IDENT
                        CORPO_CJTO_ .
180 CORPO_CJTO_     -> "," CORPO_CJTO
                    |   .
    OUTRA_CADEIA    -> ".." CADEIA
                    |   .
    OUTRO_CAR       -> ".." CARACTER
185                 |   .

    LEX_AUTOMATO    -> "automatos"
                        RGR_AUTOMATO
                    |   .
190 RGR_AUTOMATO    -> IDENT
                        "->"
                        CORPO_AUTOMAT
                        ";"
                        RGR_AUTOMATO
195                 |   .
    CORPO_AUTOMAT   -> PRIM_ELEM_AUT
                        SEG_ELEM_AUT
                        CRPO_AUTOMAT
                    |   .
200 CRPO_AUTOMAT    -> "|"
                        CORPO_AUTOMAT
                    |   .
    PRIM_ELEM_AUT   -> IDENT
                    |   CARACTER
205                 |   CADEIA .
    SEG_ELEM_AUT    -> IDENT
                    |   .

    LEX_TOKEN       -> "tokens"
210                     TOKENS_LEX .
    TOKENS_LEX      -> IDENT
                        LST_PAL_RESERV
                        ";"
                        TOKENS_LEX
215                 |   .
```

```
     LST_PAL_RESERV -> "exceto"
                       CADEIA
                       PROX_PAL_RES
                     |  .
220  PROX_PAL_RES   -> ","
                       CADEIA
                       PROX_PAL_RES
                     |  .

225  APRESENTA      -> "apresentacao"
                       APRE_CORES
                       APRE_JANELAS
                       MASCARAS
                       FIM_SECCAO
230                  |  .
     APRE_CORES     -> "cores"
                       LST_CORES
                     |  .
     LST_CORES      -> IDENT "="    /*  nome da cor    */
235                    IDENT ","    /*  cor de letra   */
                       IDENT ","    /*  cor de fundo   */
                       IDENT ","    /*  cor de borda   */
                       IDENT ","    /*  cor de sombra  */
                       IDENT ","    /*  cor de car-id  */
240                    IDENT ";"    /*  cor de off     */
                       LST_CORES_ .
     LST_CORES_     -> LST_CORES
                     |  .

245  APRE_JANELAS   -> "janela"
                       IDENTIFICACAO
                       TAM_JAN
                       ID_CORES
                       POS_JAN
250                    APRE_JANELAS_ .
     APRE_JANELAS_  -> APRE_JANELAS
                     |  .
     IDENTIFICACAO  -> IDENT
                       ";" .
255  TAM_JAN        -> "area"  "="
                       INTEIRO
                       ","
                       INTEIRO
                       ";" .
260  ID_CORES       -> "cor" "="
                       IDENT
                       ";"
                     |  .
     POS_JAN        -> JPOSIC
265                  |  .
     JPOSIC         -> "posicao" "="
                       JVAR_OU_INT
                       ","
                       JVAR_OU_INT
270                    ";" .
     JVAR_OU_INT    -> INTEIRO.
```

```
    MASCARAS        -> "mascaras"
                       LST_MASCARAS
275                 |  .
    LST_MASCARAS    -> IDENT
                       "="
                       MASCARA
                       ";"
280                    LST_MASCARAS
                    |  .
    MASCARA         -> CADEIA .

    GERACAO         -> "geracao"
285                    GER_ENTRADA
                       GER_SAIDA
                       FIM_SECCAO .
    GER_ENTRADA     -> "entrada"
                       CRPO_GER_IN .
290 CRPO_GER_IN     -> MENU
                       CRP_GER_IN_
                    |  FORM_FILL
                       CRP_GER_IN_ .
    CRP_GER_IN_     -> CRPO_GER_IN
295                 |  .

    MENU            -> "menu"
                       IDENT_GER
                       AREA
300                    CORES
                       DIR_MENU
                       POSIC_MENU
                       OPCAO_DEFAULT
                       NOME_AJUDA
305                    OPCOES_MENU .
    IDENT_GER       -> IDENT DECL_PARM ";" .
    AREA            -> "local"   "="
                       IDENT
                       ";" .
310 CORES           -> "cor"    "="
                       VAR_OU_CADEIA
                       ";" .
    DIR_MENU        -> "direcao" "="
                       DIRECAO
315                    ";" .
    DIRECAO         -> "h"
                    |  "horizontal"
                    |  "v"
                    |  "vertical"
320                 |  IDENT .
    POSIC_MENU      -> POSICAO .
    POSICAO         -> "posicao" "="
                       VAR_OU_INT
                       ","
325                    VAR_OU_INT
                       ";" .
    VAR_OU_INT      -> IDENT
                    |  INTEIRO .
    OPCAO_DEFAULT   -> "ativa" "="
```

```
330                     VAR_OU_INT ";"
                    |   .
    NOME_AJUDA      -> "ajuda"    "="
                        CADEIA
                        ";"
335                 |   .
    OPCOES_MENU     -> "opcoes"
                        OPCAO_MENU
                        OPCAO .
    OPCAO           -> OPCAO_MENU
340                     OPCAO
                    |   .
    OPCAO_MENU      -> VAR_OU_CADEIA
                        ","
                        VAR_OU_INT          /* No do caracter de   */
345                                         /* Ident.              */
                        ","
                        VAR_OU_INT          /* No de ord. da opcao */
                                            /* na visualizacao     */
                        ","
350                     ATIVO               /* Indic. de ativo/    */
                                            /* desativo            */
                        ":"
                        RET_MENU
                        NOME_AJUDA .
355 ATIVO           -> IDENT
                    |   "f" | "falso"
                    |   "v" | "verdadeiro" .
    VAR_OU_CADEIA   -> IDENT
                    |   CADEIA .
360 RET_MENU        -> IDENT_RET
                    |   CADEIA
                        ";" .
    IDENT_RET       -> IDENT
                        PARM_OPCIONAL
365                     ";" .

    FORM_FILL       -> "formulario"
                        IDENT_GER
                        AREA
370                     CORES
                        POSICAO
                        NOME_AJUDA
                        TAREFAS
                        ENTRADAS .
375 TAREFAS         -> "comandos"
                        TRFA_FF
                        LST_TAREFAS .
    LST_TAREFAS     -> TRFA_FF
                        LST_TAREFAS
380                 |   .
    TRFA_FF         -> POSIC_LABEL
                        CADEIA
                        ","
                        IDENT
385                     ":"
                        RET_FF .
```

```
     POSIC_LABEL     -> "("
                        VAR_OU_INT
                        ","
390                     VAR_OU_INT
                        ")" .                    /* POSX , POSY */
     RET_FF          -> IDENT_RET
                     |  CADEIA
                        ";" .
395  ENTRADAS        -> "entrada"
                        CARAC_ENTRADA
                        LST_ENTRADA .
     LST_ENTRADA     -> CARAC_ENTRADA
                        LST_ENTRADA
400                  |  .
     CARAC_ENTRADA   -> POSIC_LABEL
                        LABEL_NOME_CMP
                        ";"
                        NOME_AJUDA .
405  LABEL_NOME_CMP  -> TIPO_CAMPO
                     |  CADEIA
                     |  IDENT .
     TIPO_CAMPO      -> "char"
                        DESC_CAMPO
410                     DEFAULT_CHAR
                     |  "int"
                        DESC_CAMPO
                        DEFAULT_INT
                     |  "float"
415                     DESC_CAMPO
                        DEFAULT_FLOAT
                     |  "data"
                        DEFAULT_CHAR
                     |  "bool"
420                     DEFAULT_BOOL .

     DESC_CAMPO      -> TAMANHO_CAMPO
                        MASCARA_CAMPO .
     TAMANHO_CAMPO   -> "("
425                     INTEIRO
                        PRECISAO
                        ")" .
     PRECISAO        -> ","
                        INTEIRO
430                  |  .
     MASCARA_CAMPO   -> ","
                        IDENT
                     |  .
     DEFAULT_CHAR    -> "," DEF_CHAR
435                  |  .
     DEF_CHAR        -> CADEIA
                     |  IDENT .
     DEFAULT_INT     -> "," DEF_INT
                     |  .
440  DEF_INT         -> INTEIRO
                     |  IDENT .
     DEFAULT_FLOAT   -> "," DEF_FLOAT
                     |  .
```

```
    DEF_FLOAT       -> IDENT
445                    | FLOAT .
    DEFAULT_BOOL    -> "," DEF_BOOL
                       | .
    DEF_BOOL        -> "v" | "verdadeiro"
                       | "f" | "falso"
450                    | IDENT .

    GER_SAIDA       -> "saida"
                       CORPO_SAIDA
                       | .
455 CORPO_SAIDA     -> SEQ_SAIDA
                       CORPO_SAIDA_ .
    CORPO_SAIDA_    -> CORPO_SAIDA
                       | .
    SEQ_SAIDA       -> LOCAIS
460                    IDENT
                       DECL_PARM
                       "->"
                       LST_FUNCOES
                       ";" .
465 LOCAIS          -> "["
                       LST_LOCAIS
                       "]"
                       | .
    LST_LOCAIS      -> IDENT
470                    LST_LOCAIS_ .
    LST_LOCAIS_     -> ","
                       LST_LOCAIS
                       | .
    LST_FUNCOES     -> CALL_FUNC
475                    LST_FUNCOES_ .
    LST_FUNCOES_    -> LST_FUNCOES
                       | .
    CALL_FUNC       -> IDENT
                       PARM_CHAMADA .
480 PARM_CHAMADA    -> "("
                       LST_PARM_CALL
                       ")" .
    LST_PARM_CALL   -> EXPR_CHAMADA
                       LST_PARM_CALL_
485                    | .
    LST_PARM_CALL_  -> ","
                       LST_PARM_CALL
                       | .
    EXPR_CHAMADA    -> TERMO_EXPR
490                    EXPR_CHAMADA_.
    EXPR_CHAMADA_   -> ADDOP_EXPR
                       TERMO_EXPR
                       EXPR_CHAMADA_
                       | .
495 TERMO_EXPR      -> FATOR_EXPR
                       TERMO_EXPR_ .

    TERMO_EXPR_     -> MULOP_EXPR
                       FATOR_EXPR
500                    TERMO_EXPR_
```

```
                    |  .
    FATOR_EXPR      -> "("
                       EXPR_CHAMADA
                       ")"
505                 |  PARM_CALL .
    ADDOP_EXPR      -> "+"
                    |  "-" .
    MULOP_EXPR      -> "*"
                    |  "/" .
510
    PARM_CALL       -> INDIREC_VAR
                    |  ENDER_VAR
                    |  VAR_FUNC_PARM
                    |  INTEIRO
515                 |  FLOAT
                    |  CADEIA .
    INDIREC_VAR     -> "*"
                       INDIREC_VAR_ .
    INDIREC_VAR_    -> INDIREC_VAR
520                 |  IDENT .
    VAR_FUNC_PARM   -> IDENT
                       PARM_OPCIONAL .
    PARM_OPCIONAL   -> PARM_CHAMADA
                    |  .
525
    ABST_SEMAN
    LOCAL  {long posarq=0;}
                    -> "sintaxe"
                       ID_VAR_A_S
530                    CLAUSULA_GLOBAL

    /*****************************************/
                       /* reconhece a primeira vez para montar
535 */
                       /* uma tabela dos rotulos associados as
    */
                       /* producoes, com o objetivo de
    permitir*/
540                    /* a chamada dos elementos de entrada e
    */
                       /* saida no momento da traducao da des-
    */
                       /* cricao.
545 */
                       /*
    */
                       /* para poder realizar a segunda passa-
    */
550                    /* gem pelo codigo de descricao e' nes-
    */
                       /* sessario o armazenamento da posicao
    */
                       /* corrente do arquivo de descricao.
555 */

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

```
                    {posarq = 0;
                     Pos_Arquivo(&posarq);
                     posarq--; }

                    CORPO_A_S_AUX
                    FIM_SECCAO
                    FIM_GRAEDIUS_AUX


/*****************************************/
                    /* reposiciona o arquivo de descricao
*/
                    /* no  ponto armazenado e inicia a tra-
*/
                    /* ducao propriamente dita e comanda a
*/
                    /* leitura de um "token" para que o pro-
*/
                    /* cesso possa reinicializar-se.
*/
/*****************************************/

                    {Desloca_Arquivo(posarq,SEEK_SET);

Analisador_Lexico(&lin,&col,&cod,buffer);}
                    CORPO_A_S
                    FIM_SECCAO .
ID_VAR_A_S      -> "abstrata" "&" "semantica"
                 | AeS
                 | .
CLAUSULA_GLOBAL-> "global"    ":"
                    ACAO_SEMAN
                 | .
CORPO_A_S_AUX   -> PRODUCAO_AUX
                    CORPO_A_S_AUX_ .
CORPO_A_S_AUX_  -> CORPO_A_S_AUX
                 | .
PRODUCAO_AUX    -> ID_PROD_AUX
                    IDENT
                    DECL_PARM
                    LOCAL_PROD_AUX
                    ACOES_IN_PROD_AUX
                    ACOES_OUT_PROD_AUX
                    "->"
                    CORPO_SINT_AUX
                    ";" .
ID_PROD_AUX     -> "["
                    IDENT
                    "]"
                 | .
LOCAL_PROD_AUX -> "local"     ":"
                    ACAO_SEMAN
                 | .
ACOES_IN_PROD_AUX  -> "acoes_in"   ":"
```

```
615                          ACAO_SEMAN
                           |  .
     ACOES_OUT_PROD_AUX -> "acoes_out" ":"
                             ACAO_SEMAN
                           |  .
620  CORPO_SINT_AUX  -> CORPO_PROD_AUX
                        CORPO_SINT_AUX_ .
     CORPO_SINT_AUX_ -> "|"
                        CORPO_SINT_AUX
                      |  .
625  CORPO_PROD_AUX -> IDENT
                       PARM_OPCIONAL
                       CORPO_PROD_AUX
                     |  CADEIA
                       CORPO_PROD_AUX
630                  |  ACAO_SEMAN
                       CORPO_PROD_AUX
                     |  .
     FIM_GRAEDIUS_AUX -> "Fim_Def"
                           "." .
635
     CORPO_A_S       -> PRODUCAO
                        CORPO_A_S_ .
     CORPO_A_S_      -> CORPO_A_S
                      |  .
640  PRODUCAO        -> ID_PROD
                        IDENT
                        DECL_PARM
                        LOCAL_PROD
                        ACOES_IN_PROD
645                     ACOES_OUT_PROD
                        "->"
                        CORPO_SINT
                        ";" .
     ID_PROD         -> "["
650                     IDENT
                        "]"
                      |  .
     DECL_PARM       -> "("
                        LST_PARM
655                     ")"
                      |  .
     LOCAL_PROD      -> "local"     ":"
                        ACAO_SEMAN
                      |  .
660  ACOES_IN_PROD   -> "acoes_in"  ":"
                        ACAO_SEMAN
                      |  .
     ACOES_OUT_PROD -> "acoes_out" ":"
                        ACAO_SEMAN
665                   |  .
     CORPO_SINT      -> CORPO_PROD
                        CORPO_SINT_ .
     CORPO_SINT_     -> "|"
                        CORPO_SINT
670                   |  .
     CORPO_PROD      -> IDENT
```

```
                        PARM_OPCIONAL
                        CORPO_PROD
                    |   CADEIA
675                     CORPO_PROD
                    |   ACAO_SEMAN
                        CORPO_PROD
                    |   .

680 REGRAS_ASSOC    -> "regras"
                        ID_VAR_REGRA
                        {printf("\n");}
                        CORPO_REGRA
                        FIM_SECCAO .
685 ID_VAR_REGRA    -> "de" "associacao"
                    |   "associacao"
                    |   .
    CORPO_REGRA     -> PROD_ASSOC
                        CORPO_REGRA_ .
690 CORPO_REGRA_    -> CORPO_REGRA
                    |   .
    PROD_ASSOC      -> IDENT
                        "->"
                        IDENT
695                     PARM_OPCIONAL
                        ";" .

    ASSINATURAS     -> "assinaturas"
                        CORPO_ASSINAT
700                     FIM_SECCAO
                    |   .
    CORPO_ASSINAT   -> DECL_FUNC
                        CORPO_ASSINAT
                    |   .
705 DECL_FUNC       -> ESPEC_DECLAR
                        IDENT
                        DECL_PARM
                        ";" .
    ESPEC_DECLAR    -> CL_MODIF
710                     CL_TIPO
                        POINTER .
    CL_MODIF        -> CL_MOD_LOCAL
                        CL_MOD_STORAG
                        CL_MOD_SINAL
715                     CL_MOD_TAM .
    CL_MOD_LOCAL    -> "static"
                    |   "extern"
                    |   .
    CL_MOD_STORAG   -> "register"
720                 |   .
    CL_MOD_SINAL    -> "signed"
                    |   "unsigned"
                    |   .
    CL_MOD_TAM      -> "short"
725                 |   "long"
                    |   .
    CL_TIPO         -> "char"
                    |   "int"
```

```
                    |   "void"
730                 |   "float"
                    |   "double"
                    |   IDENT .
    POINTER         -> CL_MOD_POINTER
                       "*"
735                    POINTER_
                    |  .
    POINTER_        -> "*"
                       POINTER_
                    |  .
740 CL_MOD_POINTER  -> "far"
                    |  "near"
                    |  "huge"
                    |  .
    LST_PARM        -> PARAMETRO
745                    LST_PARM_
                    |  .
    LST_PARM_       -> ","
                       PARAMETRO
                    |  .
750 PARAMETRO       -> CL_CONST
                       CL_MOD_SINAL
                       CL_MOD_TAM
                       CL_TIPO
                       POINTER
755                    NOME_PARM
                       LST_PARM_
                    |  "..." .
    CL_CONST        -> "const"
                    |  .
760 NOME_PARM       -> IDENT
                    |  .

    FIM_SECCAO      -> "fim"
                       ";"
765                 |  .
```

ANEXO 8

Listagem da definição de uma interface, usando a linguagem definida para o protótipo de GRAEDIUS.

```
/*****************************************************************************/
/*                                                                           */
/*  Definicao de uma aplicacao pelo uso de GRAEDIUS.                         */
/*                                                                           */
/*     Um exemplo.                                                           */
/*                                                                           */
/*               Edson Gellert Schubert    -    CPGCC/UFRGS.               */
/*                                                                           */
/*****************************************************************************/

DEF_INTERFACE : Teste_Grafico;

/*****************************************************************************/
/*                                                                           */
/*  Nesta seccao sao declarados  :                                           */
/*                                                                           */
/*     . os conjuntos de caracteres validos para uso no reconhecedor.        */
/*                                                                           */
/*     . os automatos (Gramaticas Regulares) que definem os TOKENS usados    */
/*       pela GRAEDIUS para o reconhecimento das tarefas a executar da a-    */
/*       plicacao e os parametros necessarios `a cada tarefa.                */
/*                                                                           */
/*     . os TOKENS variaveis e os costantes. Estes ultimos aparecem na       */
/*       clausula EXCETO.                                                    */
/*                                                                           */
/*****************************************************************************/

LEXICOS

        CONJUNTOS
                Letra   -> "A".."Z","a".."z","_";
                Digito  -> "0".."9";
                Alfanum -> Digito,Letra;
                Aspas   -> #34;
                Ponto   -> #46;
                Sinais  -> #43,#45;
                ConjCda -> " ".."!","#".."}";
                IGNORE  -> EOLN," ",#01..#31;

        AUTOMATOS
                MAIS    -> #43;
                MENOS   -> #45;

                NUMERO  -> Digito Numero_ | Digito ;
```

```
45              Numero_ -> Digito NUMERO  | Digito ;

                REAL  -> Digito REAL  | Ponto Real1 ;
                Real1 -> Digito Real1 | Digito ;

50              IDENT  -> Letra Ident_  | Letra;
                Ident_ -> Alfanum Ident_ | Alfanum;

                CADEIA      -> Aspas CorpoCadeia;
                CorpoCadeia -> ConjCda CorpoCadeia | Aspas Str_Aspas | Aspas;
55              Str_Aspas   -> Aspas CorpoCadeia;

        TOKENS
             MAIS;
             MENOS;
60           NUMERO;
             REAL;
             CADEIA;
             IDENT EXCETO "arquivos",
                          "abrir",
65                        "alterar",
                          "salvar",
                          "fechar",
                          "deletar",
                          "desenhar",
70                        "circulo",
                          "reta",
                          "ponto",
                          "texto",
                          "terminar",
75                        "confirma",
                          "confirma1",
                          "cancela";

    /*****************************************************************************/
80  /*                                                                           */
    /*  Nesta seccao sao definidos os DOMINIOS e CONTRA-DOMINIOS de cada uma     */
    /*  das funcoes da biblioteca da aplicacao/linguagem utilizadas.             */
    /*                                                                           */
    /*  A declaracao dos parametros e retornos das funcoes e' feita de forma     */
85  /*  analoga `a utilizada pelo C, onde procedimentos sao funcoes que nao re-  */
    /*  tornam valor algum.                                                      */
    /*                                                                           */
    /*****************************************************************************/

90  ASSINATURAS

            int  Abre_Arquivo(char *);
            int  Alt_Arquivo(char *);
            void Salva_Arquivo(void);
95          void Fecha_Arquivo(void);
            int  Del_Arquivo(char *);

            void Circulo(float, float, float);
```

```
            void Line(float, float, float, float);
100         void Point(int, int);
            void Escreve(int, int, char *, char *);

    /*************************************************************************/
    /*                                                                       */
105 /* Nesta seccao sao declaradas informacoes que serviram de apoio para a  */
    /* correta e completa definicao da Interface do Usuario. Aqui sao decla- */
    /* rados :                                                               */
    /*                                                                       */
    /*    . conjuntos de cores que definem as cores validas para a letra, o  */
110 /*      fundo, a borda da janela, a sombra produzida pela janela, o ca-  */
    /*      racter de identificacao da opcao (no caso de MENU) e a cor da le- */
    /*      tra das opcoes/rotulos/campos desabilitados.                     */
    /*                                                                       */
    /*    . janelas (areas da tela de video) passiveis de uso, com definicao */
115 /*      da quantidade de linhas e colunas e o conjunto de cores "default" */
    /*      associado com a area.                                            */
    /*                                                                       */
    /*    . mascaras de edicao para as entradas dos parametros necessarios.  */
    /*                                                                       */
120 /*************************************************************************/

    APRESENTACAO

            CORES
125             /*  letra, fundo, borda,  sombra,ID,   off */
                c1 = black, white, blue,   black, red,  green;
                c2 = green, blue , magenta,red,   white,blue;
                c3 = black, yellow,white,  blue,  red,  magenta;
                c4 = black, aqua,  white,  blue,  red,  magenta;
130
            JANELA JMenu1;
                  AREA = 01,78; /* 01 linha e 78 colunas */
                  COR  = c1;
            JANELA JMenu2;
135               AREA = 05,10; /* 05 linha e 10 colunas */
                  COR  = c2;
            JANELA JMenu3;
                  AREA = 04,10; /* 04 linha e 10 colunas */
                  COR  = c2;
140         JANELA JMenu4;
                  AREA = 02,10; /* 02 linha e 10 colunas */
                  COR  = c2;
            JANELA JFF1;
                  AREA = 05,20; /* 04 linha e 20 colunas */
145               COR  = c2;
            JANELA JFF2;
                  AREA = 03,45; /* 03 linha e 20 colunas */
                  COR  = c3;
            JANELA JFF3;
150               AREA = 06,50; /* 05 linha e 50 colunas */
                  COR  = c4;
            JANELA JFF5;
```

```
                AREA = 05,50; /* 05 linha e 50 colunas */
                COR  = c1;
155         JANELA JFF6;
                AREA = 07,50; /* 05 linha e 50 colunas */
                COR  = c3;
            JANELA JWork;
                AREA    = 20,38; /* 20 linha e 38 colunas */
160             COR     = c3;
                POSICAO = 0,3;
            JANELA JWork1;
                AREA    = 20,38; /* 20 linha e 38 colunas */
                COR     = c4;
165             POSICAO = 40,3;

            MASCARAS
                              /*12345678901234567890*/
                 masc_1 = "__";
170              masc_2 = "__________";
                 masc_3 = "_";
                 masc_4 = "_____________";
                 masc_5 = "___._";

175
    /*****************************************************************************/
    /*                                                                           */
    /*  Nesta seccao sao definidas as formas com as quais GRAEDIUS vai realizar  */
    /*  a interacao com o usuario da aplicacao.                                  */
180 /*                                                                           */
    /*  Para que o usuario possa descrever a LINGUAGEM DE ACAO, existe a sub-    */
    /*  seccao de ENTRADA. Esta descreve os estilos de interacao, que permitem   */
    /*  a definicao dos MENUS, FORMULARIOS (FORM-FILL) e LING.COMANDO necessa-   */
    /*  rios para a especificacao das tarefas que o usuario pode executar.       */
185 /*                                                                           */
    /*  Para a descricao da LINGUAGEM DE APRESENTACAO, existe a sub-seccao de    */
    /*  SAIDA, onde sao definidas as sequencias de execucao de funcoes de al-    */
    /*  guma biblioteca (da GRAEDIUS ou da aplicacao) que permite a formacao     */
    /*  de representacoes significativas dos retornos semanticos das tarefas     */
190 /*  executadas pela aplicacao.                                               */
    /*                                                                           */
    /*  Os estilos e as sequencias de funcoes, podem receber parametros prove-   */
    /*  nientes das producoes que definem a estrutura do dialogo (ver a frente)  */
    /*  e das proprias declaracoes, no caso de encadeamento de estilos de en-    */
195 /*  trada.                                                                   */
    /*                                                                           */
    /*  P.S. :                                                                   */
    /*        So podem ser encadeados os estilos que nao necessitem de computa-  */
    /*        coes intermediarias. Caso se facam necessarias, o encadeamento e'  */
200 /*        feito na seccao de SINTAXE ABSTRATA&SEMANTICA (ver a frente).      */
    /*                                                                           */
    /*****************************************************************************/

    GERACAO
205
        ENTRADA
```

```
MENU Menu1(int hab, char *_cor);
     LOCAL   = JMenu1;
     COR     = "c1";
210  DIRECAO = h;
     POSICAO = 0,0; /*aparece em 1,1 (coord. de tela) */
     ATIVA   = 1;
     AJUDA   = "desenha";
     OPCOES
215       "Arquivos",1,1,V    : "arquivos";
          "Desenhar",1,2,hab  : "desenhar";
          "Fim",     1,3,V    : "terminar";

MENU Menu2(char *_cor);
220  LOCAL   = JMenu2;
     COR     = _cor;
     DIRECAO = v;
     POSICAO = 9,2;
     OPCOES
225       "Criar",   1,1,V : "abrir";
          "Alterar", 1,2,V : "alterar";
          "Salvar",  1,3,V : "salvar";
          "Deletar", 1,4,V : "deletar";
          "Fechar",  1,5,V : "fechar";
230
MENU Menu3(int hab);
     LOCAL   = JMenu3;
     COR     = "c2";
     DIRECAO = v;
235  POSICAO = 27,2;
     OPCOES
          "Circulo", 1,1,hab : "circulo";
          "Reta",    1,2,hab : "reta";
          "Ponto",   1,3,hab : "ponto";
240       "Texto",   1,4,hab : "texto";

MENU Menu4(char *_cor);
     LOCAL   = JMenu4;
     COR     = _cor;
245  DIRECAO = v;
     POSICAO = 35,10;
     OPCOES
          "Confirma", 1,1,TRUE : "confirma";
          "cAncela",  2,2,TRUE : "cancela";
250
formulario Formul1(char *msg, int posx, int posy);
     local   = JFF1;
     cor     = "c4";
     posicao = posx,posy; /* x e y sao inf. como parametros */
255  comandos
          (01,05) "<F10>-Conf",F10 : Menu4 ("c1")
                                     /*"confirma"*/ ;
          (01,05) "",          CR  : Menu4 ("c1")
                                     /*"confirma"*/ ;
260       (12,05) "<ESC>-Can", ESC : "cancela";
```

```
        entrada
              (02,01) msg;              /* Tit. do Form.*/
              (03,03) "[";              /* Label Puro   */
              (16,03) "]";              /* Label Puro   */
265           (04,03) char(12),masc_2,"semnome" ;
                                        /* ent. de nome */

    formulario Formul2;
        local   = JFF2;
270     cor     = "c1";
        posicao = 18,24;
        comandos
              (05,03) "<F10>-Confirma",F10 : "confirma";
              (30,03) "<ESC>-Cancela", ESC : "cancela";
275     entrada
             (02,01) "Confirma a DELECAO do arquivo (S/N) ? [ ]";
             (41,01) bool, Verdadeiro;   /* ent. da esc. */

    formulario Formul3(int posx, int posy);
280     local   = JFF3;
        cor     = "c2";
        posicao = posx,posy;   /* x e y sao inf. como parametros */
        comandos
              (05,06) "<F10>-Confirma",F10 : "confirma";
285           (30,06) "<ESC>-Cancela", ESC : "cancela";
        entrada
             (04,01) "Informe as caracteristicas do CIRCULO :";
             (04,03) "Coordenadas do centro : X = [    ]";
             (33,03) int(3),masc_1;     /* ent. de x    */
290          (38,03) "; Y = [    ]";     /* Label puro   */
             (45,03) int(3),masc_1;     /* ent. de y    */
             (04,04) "Tamanho do RAIO : [       ]";
                                        /* Label puro   */
             (23,04) float(3,1),masc_5; /* ent. do raio */
295
    formulario Formul4(int posx, int posy);
        local   = JFF3;
        cor     = "c3";
        posicao = posx,posy; /* x e y sao inf. como parametros */
300     comandos
              (07,06) "<F10>-Confirma",F10 : "confirma";
              (33,06) "<ESC>-Cancela", ESC : "cancela";
        entrada
             (04,01) "Informe as caracteristicas do RETA :";
305          (04,03) "Coordenadas do Inicio : X = [    ]";
             (33,03) int(3),masc_1;     /* ent. de x    */
             (38,03) "; Y = [    ]";     /* Label puro   */
             (45,03) int(3),masc_1;     /* ent. de y    */
             (04,04) "Coordenadas do Final  : X = [    ]";
310          (33,04) int(3),masc_1;     /* ent. de x    */
             (38,04) "; Y = [    ]";     /* Label puro   */
             (45,04) int(3),masc_1;     /* ent. de y    */

    formulario Formul5;
```

```
315                 local   = JFF6;
                    cor     = "c4";
                    posicao = 15,5; /* x e y sao inf. como parametros */
                    comandos
                          (07,05) "<F10>-Confirma",F10 : "confirma";
320                       (30,05) "<ESC>-Cancela", ESC : "cancela";
                    entrada
                         (04,01) "Informe as coordenadas do PONTO :";
                         (04,03) "X = [    ]";
                         (09,03) int(3),masc_1;     /* ent. de x    */
325                      (14,03) "; Y = [    ]";    /* Label puro   */
                         (21,03) int(3),masc_1;     /* ent. de y    */

               formulario Formul6;
                    local   = JFF6;
330                 cor     = "c1";
                    posicao = 15,5; /* x e y sao inf. como parametros */
                    comandos
                          (07,07) "<F10>-Confirma",F10 : "confirma";
                          (30,07) "<ESC>-Cancela", ESC : "cancela";
335                 entrada
                         (04,01) "Informe as caracteristicas do TEXTO :";
                         (04,03) "Coordenadas do Inicio : X = [    ]";
                         (33,03) int(3),masc_1;     /* ent. de x    */
                         (38,03) "; Y = [    ]";    /* Label puro   */
340                      (45,03) int(3),masc_1;     /* ent. de y    */
                         (28,02) "Data Atual [        ]";
                         (40,02) data,"260391";     /* data */
                         (04,04) "Qual o texto (maximo 15 caracteres) :";
                         (04,05) "[";               /* car. de inicio */
345                      (21,05) "]";               /* car. de final  */
                         (05,05) char(15), masc_4, "texto qualquer";
                                                    /* texto a escrever */

          SAIDA
350           [JWork] Circ(float x, float y, float r)
                              -> Circulo(x,y,r)
                                 Line(x-r,y,x+r,y)
                                 Line(x,y-r,x,y+r);

355           [JWork,JWork1] Reta(int x1, int y1, int x2, int y2)
                                -> Line(x1,y1,x2,y2)
                                   Circulo(x1,y1,2)
                                   Circulo(x2,y2,2);

360           [JWork] Ponto(int x1, int y1)
                              -> Point(x1,y1);

              [JWork,JWork1] Esc_Texto(int x1, int y1, char *ddata, char *cadeia)
                                       -> Escreva(x1,y1,ddata,cadeia);
365
     /*************************************************************************/
     /*                                                                       */
     /*  Nesta seccao sao definidas as relacoes existentes entre os estilos de  */
```

```
        /* interacao (de entrada) e as producoes que definem a estrutura do dia-  */
370     /* logo.                                                                   */
        /*                                                                         */
        /* A identificacao do tipo de regra que cada relacao define e' determina- */
        /* da pelo tipo de elemento relacionado com cada rotulo de producao. Se e' */
        /* um elemento definido na sub-seccao de ENTRADA da seccao de GERACAO, en- */
375     /* tao e' uma REGRA DE ENTRADA e deve ser executada antes que seu corpo    */
        /* seja reconhecido (derivado) e da avaliacao de qualquer expressao atre- */
        /* lada `a producao em questao.                                            */
        /* Caso o relacionamento seja com um elemento de SAIDA da seccao de GERA-  */
        /* CAO, entao sera uma REGRA DE SAIDA, e devera ser executada quando todo  */
380     /* o corpo da producao tiver sido reduzido (reconhecido) e todas as ex-    */
        /* pressoes tiverem sido executadas.                                       */
        /*                                                                         */
        /* Os parametros associados aos estilos/sequencias sao variaveis ativas    */
        /* (visiveis) no escopo da producao (veja uma producao como uma funcao de  */
385     /* C ou PASCAL).                                                           */
        /*                                                                         */
        /*************************************************************************/

        REGRAS
390
             inic  -> Menu1(habilita,_cor);

             arq   -> Menu2(_cor);

395          abre  -> Formul1("Arquivo a ABRIR",  5,5);
             alt   -> Formul1("Arquivo a ALTERAR",30,9);
             del   -> Formul1("Arquivo a DELETAR",55,13);
             rconf -> Formul2;

400          des   -> Menu3(hab);

             c1    -> Formul3(15,5);
             c2    -> Circ(x_c,y_c,raio);
             r1    -> Formul4(15,5);
405          r2    -> Reta(x1,y1,x2,y2);
             p1    -> Formul5;
             p2    -> Ponto(x,y);
             t1    -> Formul6;
             t2    -> Esc_Texto(x,y,ddata,cadeia);
410
        /*************************************************************************/
        /*                                                                       */
        /* Nesta seccao e' definida a estrutura do dialogo que estara a dispo-   */
        /* sicao do usuario desta aplicacao.                                     */
415     /*                                                                       */
        /* Esta aplicacao permite a criacao de retas e circulos na janela de     */
        /* trabalho (JWork).                                                     */
        /*                                                                       */
        /* De tao simples, torna-se ate' meio idiota, mas o que se busca e'      */
420     /* mostrar e explorar as caracteristicas disponiveis em GRAEDIUS.        */
        /*                                                                       */
        /*************************************************************************/
```

Sintaxe A&S

```
425
                    /************************************/
                    /*                                  */
                    /* Neste ponto devem ser declarados */
                    /* todos os procedimentos de ini-   */
430                 /* cializacao da IU.                */
                    /*                                  */
                    /* Inicializa a variavel HABILITA   */
                    /* que indica se e' acessivel a     */
                    /* opcao de desenho.                */
435                 /*                                  */
                    /************************************/

        GLOBAL : {#include <stdio.h>
                  #include <string.h>
440               #include <stdlib.h>

                  int  habilita = FALSE;
                  char _cor[]   = "cl"; }

445                 /************************************/
                    /*                                  */
                    /* Aqui comeca a gramatica que de-  */
                    /* fine a estrutura do dialogo dis- */
                    /* ponivel na aplicacao.            */
450                 /*                                  */
                    /************************************/

        [inic] INICIO -> ARQUIVOS(&habilita) INICIO
                       | DESENHAR(habilita) INICIO
455                    | "terminar" ;

             ARQUIVOS(int *des) -> "arquivos"
                                   ARQ_AUX(des) ;

460     [arq] ARQ_AUX(int *des)  -> "abrir"
                                    ABRE_ARQUIVO(des)
                                    ARQ_AUX(des)
                                  | "alterar"
                                    ALT_ARQUIVO(des)
465                                 ARQ_AUX(des)
                                  | "salvar"
                                      { Salva_Arquivo(); }
                                    ARQ_AUX(des)
                                  | "fechar"
470                                   { Fecha_Arquivo();
                                      *des = FALSE; }
                                    ARQ_AUX(des)
                                  | "deletar"
                                    DEL_ARQUIVO(des)
475                                 ARQ_AUX(des)
                                  | "cancela" ;
```

```
[abre] ABRE_ARQUIVO(int *des)
       LOCAL : {char nome[30];}
                                -> CADEIA
                                   {strcpy(nome,buffer);}
                                   ABRE_AUX(des,nome)
                                 | "cancela" ;

       ABRE_AUX(int *des, char *nome)
                               -> "confirma"
                                  {if (Abre_Arquivo(nome))
                                    *des = TRUE; }
                                 | "cancela";

[alt] ALT_ARQUIVO(int *des)
      LOCAL : {char nome[30];}
                               -> CADEIA
                                  {strcpy(nome,buffer);}
                                  ALT_AUX(des,nome)
                                | "cancela" ;

     ALT_AUX(int *des, char *nome)
                               -> "confirma"
                                  {if (Alt_Arquivo(nome))
                                    *des = TRUE; }
                                | "cancela" ;

[del] DEL_ARQUIVO(int *des)
      LOCAL    : { int confirma;
                   char nome[30]; }
                               -> CADEIA
                                  {strcpy(nome,buffer);}
                                  DEL_AUX(des,nome)
                                | "cancela" ;

     DEL_AUX(int *des, char *nome)
      LOCAL : {int confirma;}
      ACOES_IN : { confirma = FALSE; }
                               -> "confirma"
                                  RE_CONFIRMA(&confirma)
                                      {if (confirma)
                                        if (Del_Arquivo(nome))
                                         *des = FALSE; }
                                | "cancela" ;

[rconf] RE_CONFIRMA(int *conf) -> "confirma"
                                  IDENT
                                    { if (!strcmp(buffer,"S"))
                                       *conf = TRUE;
                                      else
                                       *conf = FALSE; }
                                | "cancela"
                                  IDENT
                                    { *conf = FALSE; } ;
```

```
        DESENHAR (int hab) -> "desenhar" DES_AUX(hab) ;

      [des] DES_AUX(int hab)
535         LOCAL    : { int x,y; char *aux; }
                    -> "circulo"
                       DES_CIRCULO
                       DES_AUX(hab)
                     | "reta"
540                    { x = 13;
                         y = 4;
                         if ((aux = (char *) calloc(1,80)) == NULL)
                         BEGIN
                          error_message("FATAL : problemas na alocacao de memoria \"DES_AUX\".");
545                       exit(44);
                         END
                         aux[0] = 0;
                         sprintf(aux,"Vai mostrar o formulario em x=%d,y=%d.",
                                 x,y);
550                      message(aux);
                         free(aux); }
                       DES_RETA
                       DES_AUX(hab)
                     | "ponto"
555                    DES_PONTO
                       DES_AUX(hab)
                     | "texto"
                       DES_TEXTO
                       DES_AUX(hab)
560                  | "cancela" ;

      [c1] DES_CIRCULO -> CONF_CIRC_AUX
                        | CANC_DESENHO ;

565       CANC_DESENHO
            ACOES_OUT : { printf("%c",7); }
                      -> "cancela" ;

      [c2] CONF_CIRC_AUX
570         LOCAL     : { int x_c,y_c,sinal; float raio; }
            ACOES_IN  : { x_c = y_c = 15; raio = 5.0; }
                      -> "confirma"
                         SINAL(&sinal)
                         NUMERO
575                      { x_c  = atoi(buffer);
                           x_c *= sinal; }
                         SINAL(&sinal)
                         NUMERO
                         { y_c  = atoi(buffer);
580                        y_c *= sinal; }
                         SINAL(&sinal)
                         CONF_CIRC_1(&raio)
```

```
585         SINAL (int *sinal) -> MENOS { *sinal = -1; }
                                | MAIS  { *sinal =  1; } ;

            CONF_CIRC_1(float *raio, int sinal)
                                    -> REAL
590                                    { *raio = atof(buffer);
                                         *raio *= sinal; }
                                    | NUMERO
                                      { *raio = atoi(buffer) * 1.0;
                                        *raio *= sinal; } ;
595
        [r1] DES_RETA -> CONF_RETA_AUX
                       | CANC_DESENHO ;

        [r2] CONF_RETA_AUX
600           LOCAL    : { int x1,y1,x2,y2,sinal; }
              ACOES_IN : { x1 = y1 = 0; x2 = y2 = 25; }
                         -> "confirma"
                            SINAL(&sinal)
                            NUMERO
605                         { x1 = atoi(buffer);
                              x1 *= sinal; }
                            SINAL(&sinal)
                            NUMERO
                            { y1 = atoi(buffer);
610                           y1 *= sinal; }
                            SINAL(&sinal)
                            NUMERO
                            { x2 = atoi(buffer);
                              x2 *= sinal; }
615                         SINAL(&sinal)
                            NUMERO
                            { y2 = atoi(buffer);
                              y2 *= sinal; }

620     [p1] DES_PONTO -> CONF_PTO_AUX
                       | CANC_DESENHO ;

        [p2] CONF_PTO_AUX
              LOCAL    : { int x,y,sinal; }
625           ACOES_IN : { x = y = 15; }
                         -> "confirma"
                            SINAL(&sinal)
                            NUMERO
                            { x = atoi(buffer);
630                           x *= sinal; }
                            SINAL(&sinal)
                            NUMERO
                            { y = atoi(buffer);
                              y *= sinal; }
635
        [t1] DES_TEXTO -> CONF_TXT_AUX
                       | CANC_DESENHO ;
```

```
        [t2] CONF_TXT_AUX
640             LOCAL    : { int  x,y,sinal;
                             char cadeia[40] = "Texto a imprimir",
                                  ddata[8]; }
                ACOES_IN : { x = 3; y = 1; }
                           -> "confirma"
645                           SINAL(&sinal)
                              NUMERO
                              { x = atoi(buffer);
                                x *= sinal; }
                              SINAL(&sinal)
650                           NUMERO
                              { y = atoi(buffer);
                                y *= sinal; }
                              CADEIA
                              { strcpy(ddata,buffer); }
655                           CADEIA
                              { strcpy(cadeia,buffer); }

    FIM_DEF .
```

BIBLIOGRAFIA

[AHO 86] AHO,Alfred;SETHI,Ravi;ULLMAN,Jeffrey D. Compilers : principles, techniques and tools. Reading, Addison-Wesley, 1986.

[BAG 89] BAGGIO,A. Uma interface de gerenciamento para ao projeto TRANCA. Porto Alegre, CPGCC/UFRGS, 1989. Proj. Dipl.

[BEN 86] BENNETT,J.L. Tools for building advanced user interfaces. IBM Systems Journal, Armonk, 25(3/4) : 354-368, july 1986.

[BET 87] BETTS,Bill;BURLINGAME,David;FISCHER,Gerhard;FOLEY, Jim;GREEN,Mark;KASIK,David;KERR,Stephen T.; OLSEN,Dan;THOMAS,James. Goals and objetives for user interface software. Computer Graphics, New York, 21(2):73-78, Apr. 1987.

[BOR 86] BORLAND INTERNATIONAL. Turbo PROLOG : owner's handbook. Scotts Valley, 1986. 224p.

[BOR 88a] BORLAND INTERNATIONAL. Turbo C : owner's handbook. Scotts Valley, 1988.

[BOR 88b] BORLAND INTERNATIONAL. Turbo PASCAL : owner's handbook. Scotts Valley, 1988.

[COU 85] COUTAZ,Joëlle. Abstractions for user interface design. Computer, Los Angeles, 18 : 21-34, Sept. 1985.

[DAT 86] DATE,C.J. An introduction to database systems. Reading, Addison-Wesley, 1986.

[ESP 89] ESPERANÇA,Lúcia G. Um interpretador de gramáticas de atributos. Porto Alegre, CIC/UFRGS, Dec. 1989. Proj. de Dipl.

[FAV 87] FAVERO,Eloi Luiz. Um editor orientado por estruturas para linguagens diagramáticas. Porto Alegre, CPGCC da UFRGS, 1989. Diss. de Mest.

[FIS 88] FISHER,Alan S. User Interface Design Methodologies. In : CASE : using software development tools. New York, John Wiley & Sons Inc., 1988. p. 117-137.

[FOL 82] FOLEY,James D.;Van DAM,Andreis. Fundamentals of interactive computer graphics. Reading, Addison-Wesley. 1982. 664p.

[GAN 83] GANE,Chris. Análise estruturada de sistemas. Livros Técnicos e Científicos, Rio de Janeiro, 1983.

[GRE 86] GREEN,Mark. A survey of three dialogue models. ACM Transactions on Graphics, New York, 5(3) : 244-275, July 1986.

[HAR 89] HARTSON,H. Rex;HIX Deborah. Human-computer interface development : concepts and systems for its management. Computing Surveys, New York, 21(1) : 5-92, March 1989.

[HOR 86] HORWITZ,Susan;TEITELBAUM,Tim. Generating editing environment based on relations and attributes. ACM Transactions on Programing Languages and Systems, 8(4) : 577-608, Oct. 1986.

[HUD 86] HUDSON,Scott E.;King,Roger. A generator of direct manipulation office systems. ACM Transaction on Office Information Systems, New York, 4(2) : 132-163, Apr. 1986.

[HUT 86] HUTCHINS,E.L.;HOLLAN,J.D.;NORMAN,D.A. Direct Manipulation Interfaces. In : USER centered system design, Hillsdale, Lawrence Erlbaum Assoc., 1986, p. 87-124.

[MCC 89] McCLURE,Carma. Habitable Environments. In : Case is software automation. Englewood Cliffs, Prentice Hall, 1989, p. 215-243.

[MOR 81] MORAN,Thomas P. The command language grammar : a representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies, Londres, 15 : 3-50, MES 1981.

[MYE 89] MYERS,Brad A. User interface tools : introduction and survey. IEEE Software, Los Alamitos, 6(1), p.15-23, Jan. 1989.

[NOR 88] NORTON,Peter. Desvendando o PC : Acesso a caracteristicas avançadas e programaçâo. Rio de Janeiro, Campus, 1988. 239p.

[OAK 86] OAKEY,Steve. LISP para Micros. Rio de Janeiro, Campus, 1986. 186p.

[OLS 86] OLSEN Jr.,Dan R. MIKE : the menu interaction kontrol environment. ACM Transactions on Graphics, New York, 5(4) : 318-344, Oct. 86.

[PIM 91] PIMENTA,Marcelo S. Um modelo canônico de ferramenta para desenvolvimento de interface com o usuário. Porto Alegre, CPGCC/UFRGS, 1991. Dis. Mestrado.

[RHY 86] RHYNE,Jim;EHRICH,Roger;BENNETT,John;HEWTT,Tom; SIBERT,John;Bleser,Terry. Tools and methodology for user interface development. Computer Graphics, New York, 21(2):78-86, Apr. 1987.

[ROS 89] ROSA,Fernando Raupp. SINLEX : um ambiente de desenvolvimento de processadores de linguagens. Porto Alegre, CPGCC/UFRGS, 1989. Proj. Dipl.

[SCH 90] SCHUBERT,Edson G.,PIMENTA,Roberto T. GRAEDIUS : uma proposta para definição de interfaces baseada em gramática de atributos. In : Simpósio Brasileiro de Engenharia de Software, 4o, 1990, Águas de São Pedro, Anais, São Paulo : USP, 1990. 280p., p.84-95.

[SHN 86] SHNEIDERMAN,Ben. Designing the user interface : strategies for effective human-computer interaction. Reading, Addison-Wesley, 1986.

[SIB 86] SIBERT,John L.;HURLEY,Willian D.;BLESER,Teresa W. An object-oriented user interface management system. Computer Graphics, New York, 20(4) : 259-268, Aug. 1986.

[SUN 89] SUN MICROSYSTEMS. Sun system user's guide, 1989.

[TAK 89] TAKAHASHI,Tadao. Introdução a programação orientada a objetos. Curitiba, EBAI, 1988. 148p. Apresentado na III EBAI, Curitiba, 1988.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"Uma linguagem de definição e manipulação de interfaces com o usuário".

Dissertação apresentada aos Srs.

Prof. Dr. Carlos Alberto Heuser

Prof. Dr. Paulo Cesar Masiero (ICMSC-USP)

Prof. Dr. Raul Fernando Weber

Prof. Dr. Roberto Tom Price

Visto e permitida a impressão

Porto Alegre, 30 / 10 / 91

Prof. Dr. Roberto Tom Price
Orientador

Prof. Dr. Ricardo Augusto da L. Reis
Coordenador do Curso de Pós-Graduação
em Ciência da Computação

ESR/CCB

\DIVCPGCC\ASSINA.BAN