SISTEMAS PARA O DESENVOLVIMENTO DE INTERFACES DE USUÁRIO

Roberto Carlos Mayer

Dissertação apresentada ao
Departamento de Ciência de Computação
do Instituto de Matemática e Estatística
da Universidade de São Paulo
para a obtenção do grau de
Mestre em Matemática Aplicada

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Valdemar W. Setzer

São Paulo, Brasil

Dezembro de 1992

*Ao meu filho Bruno,
pelas horas de convívio sacrificadas
nos seus primeiros meses de vida
para a conclusão deste trabalho.*

AGRADECIMENTOS

Ao Prof. Dr. Valdemar Waingort Setzer, pela orientação deste trabalho, pela atenção e paciência que possibilitaram a elaboração desta dissertação e pelos valiosos conselhos recebidos;

Ao Prof. Dr. Arnaldo Mandel, como orientador de programa quando do meu ingresso no programa de mestrado do MAC;

À minha esposa, Celia, pelo constante incentivo para o prosseguimento dos meus estudos de pós graduação;

A todos os colegas e funcionários do IME, especialmente aos colegas e amigos do MAC;

E a todos aqueles que, diretamente ou indiretamente, contribuíram para que este trabalho fosse possível.

RESUMO

Este trabalho objetiva dar uma visão geral do conhecimento existente sobre interfaces de usuário e apresentar, em particular, uma ferramenta de interface de usuário original, desenvolvida pelo autor.

Uma perspectiva histórica do papel das interfaces de usuário define o contexto para o restante do trabalho. São apresentadas as características comuns e desejáveis das interfaces de usuário modernas e a base formal utilizada pela maioria das ferramentas existentes.

O trabalho inclui uma descrição das principais ferramentas comerciais e de pesquisa já desenvolvidas, e apresenta e discute detalhadamente uma ferramenta chamada **MITUI**®, desenvolvida no Brasil pelo autor.

O trabalho encerra com algumas breves considerações sobre outras facetas das interfaces de usuário, algumas conclusões sobre o estágio do conhecimento sobre interfaces de usuário e uma avaliação da ferramenta **MITUI**®.

ABSTRACT

This text aims to give a broad overview of the existing knowledge about user interfaces and, specifically, to present an original user interface tool developed by the author.

A historical perspective of the user interface's role defines the context for the remainder of the text. The common and desirable properties of modern user interfaces are described together with the formal base used by most existing tools.

The text includes a description of the main commercial and research tools available, and presents and discusses in detail a tool called **MITUI**®, developed by the author in Brazil.

The work closes with some brief pondering of other user interface related aspects, some conclusions about the present knowledge about user interfaces, and an evaluation of **MITUI**®.

ÍNDICE

CAPÍTULO 0

INTRODUÇÃO	1
0.1. As interfaces de usuário como problema	1
0.2. Vantagens do uso de ferramentas	4

CAPÍTULO 1

UMA PERSPECTIVA HISTÓRICA	7
1.1. A evolução do computador	7
1.2. A evolução da interface do computador	9
1.2.1. A era do hardware	9
1.2.2. A era do software	9
1.2.3. A era do terminal	11
1.2.4. A era do diálogo	11
1.2.5. A era do grupo	12

CAPÍTULO 2

CARACTERÍSTICAS DAS INTERFACES DE USUÁRIO	14
2.1. Realimentação	15
2.2. Disponibilidade de ajuda para o aprendizado	16
2.3. Erros do usuário e reversão	19
2.4. Tempo de resposta	20

2.5. Consistência.....	21
2.6. Visual estruturado	23
2.7. Memorização mínima.....	24
CAPÍTULO 3	
FORMALIZAÇÃO DAS INTERFACES DE USUÁRIO	26
3.1. Introdução aos modelos de diálogo	29
3.1.1. Redes de Transição.....	30
3.1.2. Gramáticas Livres de Contexto	33
3.1.3. Eventos	35
3.2. Definição formal dos modelos de diálogo	42
3.2.1. Redes de Transição.....	42
3.2.2. Gramáticas Livres de Contexto	45
3.2.3. Eventos	48
3.3. Poder de expressão.....	53
3.3.1. Conversão de Redes de Transição em Sistemas de Eventos	57
3.3.2. Conversão de Gramáticas Livres de Contexto em Sistemas de Eventos	63
3.3.3. Análise dos Algoritmos de Conversão	68
3.3.4. Consequências práticas	69
CAPÍTULO 4	
FERRAMENTAS PARA O DESENVOLVIMENTO DE INTERFACES DE USUÁRIO	70
4.1. Tipos de ferramentas de interface	71
4.2. Comunicação entre ferramenta e aplicação	72
4.3. Tipos de usuários dos UIDSs	74
4.4. User Interface Development Kits	74
4.5. User Interface Development Systems	76

4.5.1. Funções de um UIDS	76
4.5.2. Componentes de um UIDS.....	77
4.6. UIDSs baseados em linguagens	78
4.6.1. Hierarquias de menus	78
4.6.2. Redes de transição	79
4.6.3. Gramáticas livres de contexto	80
4.6.4. Linguagens de tratamento de eventos	81
4.6.5. Linguagens declarativas	83
4.6.6. Linguagens orientadas a objetos.....	84
4.7. UIDSs baseados em especificações gráficas	84
4.8. UIDSs com geração automática	87
4.9. Problemas das ferramentas de interface.....	88
 CAPÍTULO 5	
MITUI, UM UIDS BRASILEIRO	92
5.1. O padrão CUA e o Windows	92
5.2. MITUI como UIDS.....	94
5.3. Características da Interface	94
5.3.1. Realimentação	96
5.3.2. Sistema de ajuda e mensagens de prontidão	96
5.3.3. Reversão de Erros.....	99
5.3.4. Tempo de resposta	100
5.3.5. Consistência	100
5.3.6. Visual estruturado.....	101
5.3.7. Comandos fornecidos por MITUI para a aplicação.....	103
5.4. Técnicas de interação implementadas.....	105
5.4.1. Menus	106
5.4.2. Caixas de Diálogo.....	108
5.4.3. Janelas de Documento.....	110

5.4.4. Barras de Deslocamento	112
5.4.5. Macros	113
5.5. Programação com MITUI	113
5.5.1. Uma Aplicação Vazia	113
5.5.2. Uma Aplicação com Menus e Janelas	117
5.6. A Implementação de MITUI	123
5.6.1. Módulos de baixo nível	126
5.6.2. Módulos de tratamento de eventos	130
5.6.3. Outros módulos fonte	132
5.7. Uma avaliação da implementação	133
5.8. Extensões previstas para MITUI	137
5.8.1. Orientação a objetos	137
5.8.2. Ambientes gráficos	139
5.8.3. Gerador interativo de interfaces	140
CAPÍTULO 6	
OUTROS ASPECTOS SOBRE INTERFACES DE USUÁRIO	141
6.1. Atualização de interfaces já existentes	141
6.2. As interfaces, o ensino e a sociedade	142
CAPÍTULO 7	
CONCLUSÕES	144
BIBLIOGRAFIA	146

CAPÍTULO 0

INTRODUÇÃO

O termo "Interface de Usuário" é difícil de ser definido com precisão. Adotamos neste trabalho a definição apresentada em [MYERS 91], segundo a qual a interface de usuário é o *componente de software* de uma aplicação que *traduz as ações do usuário em uma ou mais solicitações à funcionalidade da aplicação* e que *fornece ao usuário realimentação** sobre as consequências da sua ação.

Portanto, esse componente do software deve ser considerado em separado do processo computacional ou algorítmico subjacente à funcionalidade da aplicação. A parte da aplicação destinada a gerar saída impressa *não* é considerada como parte da interface de usuário, já que ela é, em geral, demorada demais para servir como canal de realimentação para o usuário.

0.1. As interfaces de usuário como problema

A criação de boas interfaces de usuário é uma tarefa difícil. Não existem métodos ou técnicas que garantem que uma determinada interface de usuário seja fácil de usar, ou fácil de aprender. A grande maioria dos

---

* "Feedback", em inglês.

programadores é incapaz de criar interfaces consideradas agradáveis pelos usuários.

A implementação das interfaces de usuário é intrinsecamente difícil e cara. Um fator de complicação é que toda interface de usuário deve gerenciar uma variedade de dispositivos diferentes, como por exemplo teclado, mouse e vídeo, que podem gerar sequências de eventos de maneira assíncrona. Outra dificuldade é a necessidade intrínseca de um bom desempenho, para que a reação da aplicação às ações do usuário pareça imediata.

O código correspondente à interface de usuário é freqüentemente grande, complexo e difícil de ser depurado e modificado. A interface pode ser responsável por uma parte muito significativa do código total de uma aplicação. Por exemplo, nas aplicações de inteligência artificial (de acordo com [BOBROW 86]), 40 a 50% do total do código e da utilização de memória são destinados à interface de usuário.

À medida que as interfaces se tornam mais fáceis de usar, elas se tornam cada vez mais difíceis de criar, já que a arquitetura interna dos computadores permanece basicamente a mesma. As interfaces modernas, baseadas em equipamentos gráficos, com a utilização de múltiplos aplicativos e múltiplas janelas em cada aplicativo, estão entre as mais difíceis de serem implementadas. Chamaremos de interfaces de *manipulação direta* àquelas que permitem ao usuário agir diretamente sobre objetos visíveis na tela com ações incrementais, rápidas e reversíveis.

A criação de interfaces de manipulação direta é difícil porque é necessário fornecer ao usuário gráficos elaborados, tornando disponível cada um dos recursos de muitas maneiras diferentes, permitir a operação de vários dispositivos de entrada de forma assíncrona e concorrente, interagir com o usuário de forma *não modal* (o que significa, de maneira simplificada, que o

usuário pode usar qualquer recurso da aplicação praticamente a qualquer momento*), e uma realimentação semântica rápida.

A exigência de realimentação semântica significa que a aplicação deve reagir às ações do usuário de acordo com informação especializada a respeito dos objetos sendo manipulados pelo programa. Por exemplo, se uma opção de uma determinada caixa de diálogo se tornar não disponível por algum motivo, esse fato deve ser indicado ao usuário através de uma mudança do estado da opção (por exemplo, a cor dessa opção pode ser modificada para um tom cinzento, de visualização mais difícil que a cor usada para as opções disponíveis).

Além da dificuldade de se criar interfaces como as descritas, não existe nenhuma estratégia de projeto que garanta que a interface resultante seja fácil de aprender ou fácil de usar. De acordo com [SWART 82], a única maneira confiável de gerar interfaces de boa qualidade é através da realização de testes com protótipos da interface e da modificação do projeto em função dos comentários dos usuários .

Este método, denominado *projeto iterativo*, foi usado para criar algumas das melhores interfaces atualmente disponíveis. Por exemplo, um sistema de correio eletrônico descrito em [GOOD 84], baseado em comandos, foi melhorado até reconhecer de forma correta 76% dos comandos gerados por usuários inexperientes, enquanto a versão inicial do mesmo sistema reconhecia apenas 7% desses comandos.

---

* Denomina-se *modo* de uma interface a um estado que o usuário deve selecionar explicitamente para poder utilizar determinada parte da funcionalidade da aplicação. Por exemplo, muitos editores de texto, como o vi do Unix e o Microsoft Word 5.0 possuem um *modo de edição* e um *modo de comando* (para o usuário efetuar operações que não a edição direta do texto).

No entanto, de acordo com [JACOB 86], as interfaces de manipulação direta são altamente modais, sendo que o modo corrente é simultaneamente definido e indicado pela posição do cursor e/ou do dispositivo apontador.

A necessidade do projeto iterativo, de custo relativamente elevado, e a dificuldade de obter uma medida da qualidade de uma interface de usuário são fatores determinantes para o grande interesse existente no desenvolvimento e uso de sistemas que ajudem a projetar e implementar interfaces de usuário.

Esses sistemas-ferramentas são geralmente denominados *Sistemas de Gerenciamento de Interfaces de Usuário*, abreviado UIMS (do inglês, "User Interface Management Systems"). No entanto, concordamos com [HILL 86] e [MYERS 89], em designá-los *Sistemas para o Desenvolvimento de Interfaces de Usuário*, abreviado UIDS (do inglês, "User Interface Development Systems"), levando-se em conta o escopo das ferramentas atualmente disponíveis.

0.2. Vantagens do uso de ferramentas

O uso de ferramentas de interface de usuário apresenta uma série de vantagens, divididas basicamente em duas áreas. A primeira área diz respeito à produção de interfaces de usuário de melhor qualidade:

- Os projetos das interfaces de usuário podem ser prototipados e implementados rapidamente, possivelmente antes mesmo que o código da aplicação seja escrito.
- A facilidade de alterações da interface beneficia o processo de introdução de modificações cuja necessidade tenha sido verificada através de testes com usuários.
- Dependendo da ferramenta utilizada, uma mesma aplicação pode ter várias interfaces de usuário diferentes.

- O investimento de uma quantidade maior de tempo e dinheiro na produção de ferramentas de interface de usuário é mais viável do que na produção de uma interface para uma única aplicação, já que as ferramentas podem ser reaproveitadas em várias aplicações.
- Aplicações diferentes criadas com as mesmas ferramentas de interface de usuário têm interfaces mais consistentes entre si.
- A pesquisa de estilos diferentes para a interface de uma aplicação específica fica facilitada.
- Torna-se possível o envolvimento de especialistas, como artistas gráficos e psicólogos, no projeto de interfaces. A responsabilidade pelo projeto como um todo pode caber a projetistas profissionais de interfaces que não sejam necessariamente programadores.

A outra área onde são obtidas vantagens com o uso de ferramentas está relacionada com o código fonte da aplicação, que fica mais fácil de ser criado e mais econômico de ser mantido.

- O código correspondente à interface de usuário resulta melhor estruturado e mais modular pelo simples fato de ter sido separado do código específico da aplicação. Isto permite que o projetista modifique a interface sem precisar modificar a aplicação, e permite ao programador modificar a aplicação sem modificar a interface.
- A confiabilidade da interface resulta maior porque o código é criado automaticamente a partir de uma especificação de alto nível.
- As especificações de interface podem ser representadas, validadas e avaliadas com maior facilidade.
- As dependências dos dispositivos físicos disponíveis são isolados pelas ferramentas de interface, tornando mais fácil o porte das aplicações para ambientes diferentes.

Este trabalho objetiva dar uma visão geral do conhecimento acumulado sobre interfaces de usuário e apresentar, em particular, uma ferramenta de interface de usuário original, desenvolvida pelo autor.

O Capítulo 1 deste trabalho traça uma perspectiva histórica para o papel das interfaces de usuário, definindo o contexto para o restante do trabalho. O Capítulo 2 descreve as características comuns e desejáveis das interfaces de usuário modernas de boa qualidade, enquanto o Capítulo 3 apresenta as principais formalizações da interface de usuário utilizadas nos UIDS existentes, e a relação entre essas formalizações.

O Capítulo 4 descreve as características marcantes dos UIDS comerciais e de pesquisa já desenvolvidos. Já o Capítulo 5 apresenta e discute o UIDS comercial denominado MITUI®, desenvolvido por uma empresa da qual o autor é sócio-diretor.

O Capítulo 6 apresenta algumas breves considerações sobre outras facetas das interfaces de usuário, como a atualização de interfaces de software já existente e o ensino sobre interfaces de usuário. O trabalho encerra com algumas conclusões sobre o estágio do conhecimento sobre interfaces de usuário e uma avaliação da ferramenta MITUI®.

CAPÍTULO 1

UMA PERSPECTIVA HISTÓRICA

O termo "Interface de Usuário" é ironicamente centrado na tecnologia (enquanto o "usuário" é especificado, o computador é implícito). Para analisar o surgimento e a evolução desse conceito é conveniente pensar na "interface do computador" com o seu usuário e o mundo. Esta perspectiva, sugerida em [GRUDIN 90], permite estender o termo "interface de usuário" para o período anterior à existência do termo e também para o futuro, no qual o computador terá uma interface abrangendo não apenas usuários individuais, mas grupos e organizações. Várias partes deste Capítulo são baseadas no artigo [GRUDIN 90].

1.1. A evolução do computador

Há vinte anos, o desenvolvimento dos computadores era dominado pelo hardware. As maiores empresas de computação produziam hardware, e sua sobrevivência dependia exclusivamente do seu sucesso na venda de hardware. A medida básica para a avaliação de novos produtos consistia de análises comparativas de desempenho.

Esta situação mudou no final da década de 70 e início da década de 80, com o sucesso das planilhas de cálculo, processadores de texto e o surgimento dos sistemas operacionais não proprietários (sistemas

operacionais não desenvolvidos para uso em um equipamento específico, como por exemplo o Unix).

Esses tipos de software continuaram a aumentar as vendas de hardware de empresas como IBM, Wang, Apple e Digital. Estas empresas continuaram a ser as empresas de computação com as maiores margens de lucro.

As inovações na área de hardware continuam: até hoje muitas empresas de computação ainda concorrem com produtos de hardware, mas aceitando uma forte concorrência internacional e margens de lucro cada vez menores. Existe um grupo de empresas novas, fundadas na década de 80, como Microsoft, Lotus, Borland, Novell e outras, dedicadas primordialmente ao desenvolvimento e comercialização de software, e que são atualmente as empresas de computação que conseguem atingir as maiores margens de lucro.

Nos últimos anos iniciou-se uma nova tendência: o destaque da interface de usuário. A interface gráfica de usuário do Macintosh inicialmente gerou lucros adicionais para a Apple, com a venda de mais hardware proprietário, mas todas as empresas de software que foram atraídas pela interface acabaram ampliando o mercado de software. O ambiente gráfico Windows da Microsoft contribuiu para acelerar este processo. A importância da interface de usuário é maior nas categorias de software mais tradicionais, que apresentam produtos maduros (como por exemplo as planilhas de cálculo e os processadores de texto), já que permite criar um diferencial para os produtos, incitando os usuários a atualizar as suas versões.

1.2. A evolução da interface do computador

1.2.1. A era do hardware

Os usuários dos primeiros computadores eram engenheiros e programadores, que precisavam ter um conhecimento significativo do hardware para poder usá-lo. Eles trabalhavam usando o sistema de numeração binário, lidavam diretamente com registradores e posições de memória, e tinham como preocupação principal a maximização do desempenho do hardware, independentemente de estarem envolvidos em computação científica ou comercial. Frequentemente, era necessário o sacrifício de características desejáveis dos programas para torná-los suficientemente pequenos para caber na limitada memória disponível naqueles computadores. A mesma situação repetiu-se na década de 70 com os primeiros computadores pessoais.

1.2.2. A era do software

A interface de usuário desses equipamentos evoluiu de três maneiras. Primeiramente foi desenvolvido hardware mais confiável e veloz; segundo, a apresentação do hardware tornou-se mais ergonômica, e por último, a interface do usuário-programador foi melhorada libertando o usuário das restrições do hardware, através do surgimento das linguagens de programação de alto nível e da memória virtual, entre outros recursos.

Ao longo de toda a década de 60 e até durante a primeira metade da década de 70, os programadores continuaram a ser os principais usuários de computadores. Os primeiros terminais de vídeo custavam mais de 10.000 dólares. Os primeiros editores de texto foram criados para uso dos

programadores. O uso de computadores e terminais de vídeo para processamento de texto genérico era economicamente inviável.

Desta maneira, as melhorias na interface de usuário eram na realidade melhorias na interface de programação. Ferramentas como sistemas operacionais, compiladores e depuradores, bancos de dados, e posteriormente as técnicas chamadas de "engenharia de software" contribuíram para melhorar a produtividade dos usuários-programadores.

No entanto deve ser notado que durante este período a preocupação em facilitar o trabalho dos programadores era apenas uma das motivações. Tipicamente, *não* havia preocupação com quaisquer fatores humanos, como por exemplo a legibilidade dos programas. Mesmo no caso das linguagens de programação, o objetivo principal das pesquisas da época era o uso de abstrações de nível mais alto. Somente no final deste período é que começou a haver uma preocupação com fatores como a decomposição dos programas em módulos, sua documentação efetiva e a definição de estratégias para a depuração de programas.

Um sintoma claro dessa situação é que na conferência de 1982 efetuada em Gaithersburg, USA, que levou a criação do SIGCHI (Special Interest Group on Computer-Human Interfaces) dentro da ACM (Association for Computer Machinery), a grande maioria dos artigos apresentados ainda tratava de problemas específicos de programação (veja [GAITH 82]).

O desenvolvimento de hardware mais significativo do período foi sem dúvida o terminal de vídeo. Ele permitiu que, pela primeira vez, usuários não programadores tivessem acesso direto aos computadores. As pesquisas apresentadas nos foros do SIGCHI passaram a ser majoritariamente voltadas a ajudar os programadores a criar interfaces melhores para usuários não programadores (chamados de "usuários finais") ao invés de ajudar os programadores como usuários.

1.2.3. A era do terminal

Na etapa seguinte do desenvolvimento, a partir da segunda metade da década de 70, aparece o termo "interface de usuário" pela primeira vez. Inicialmente foram estudados fatores ergonômicos e motores, tais como a legibilidade dos terminais, o projeto de teclados e outros dispositivos de entrada (tablets, canetas óticas e mouses).

Quase que ao mesmo tempo, foram sendo introduzidos elementos além da percepção e capacidade motora do usuário, como por exemplo, o uso de teclas de função, linguagens de comandos e menus, entre outros. A pesquisa no uso destes recursos é muito menos propícia à utilização de experiências reprodutíveis, criando oportunidades para contribuições de psicólogos, por exemplo através do estudo da formação de modelos mentais e da memorização da semântica associada às interfaces de usuário.

Esse tipo de estudos ainda é necessário, na medida em que a disseminação de recursos como o uso de cores, telas gráficas e de maior tamanho, execução multitarefa e a utilização de som estéreo criam novos problemas. Nesta fase é necessária, por exemplo, a participação de artistas gráficos para projetar e avaliar as interfaces resultantes.

Em resumo, o escopo das pesquisas voltou-se completamente para o "usuário final" e, embora os processos cognitivos subjacentes à utilização de menus, layout de janelas e outras escolhas de projeto semelhantes não estejam totalmente claros, existe um certo grau de convergência entre os estilos das interfaces gráficas emergentes.

1.2.4. A era do diálogo

No período seguinte, já na década de 80, a pesquisa de ponta em interfaces de usuário voltou-se para o estudo de modelos de interface que permitissem acompanhar o trabalho do usuário, armazenando as ações

passadas de maneira a permitir a adaptação ou personalização da interface às necessidades de cada indivíduo.

Tornou-se necessário estudar o diálogo do usuário com o computador de um ponto de vista psicolingüístico. Como o nível de desenvolvimento da ciência não é suficientemente forte nessa área, existem algumas propostas de se aproveitar até o conhecimento acumulado nas artes dramáticas.

Os avanços tecnológicos a serem incorporados no diálogo incluem a disponibilidade de memória em grandes quantidades, a representação do conhecimento dentro do software e a disponibilidade ampla de sistemas operacionais gráficos e multitarefa.

É importante notar que os paradigmas de pesquisa não estão firmemente estabelecidos nesta área. O estudo dos diálogos não pode ser baseado em experiências com tempos e erros mensuráveis, mas deve ser baseado na gravação e posterior análise dos diálogos. Para tanto, o usuário pode ser filmado, ou pode ser usado um sistema de computador com respostas simuladas (um pesquisador cria as respostas sem que o usuário tenha conhecimento da existência desse pesquisador). Conseqüentemente, o avanço nas pesquisas nesta área é relativamente mais lento.

1.2.5. A era do grupo

O próximo passo na evolução da interface de usuário deverá ser a sua adaptação a grupos de indivíduos, equipes de trabalho ou com algum vínculo social. Redes locais, redes cooperativas (como a Internet) e redes corporativas são a base tecnológica deste próximo passo.

O termo "groupware" já vem sendo utilizado para software destinado a correio eletrônico, editores que permitem coautoria de textos, gerenciamento distribuído de projetos e auxílio à tomada coletiva de decisões.

No entanto, o progresso alcançado nesta área ainda é muito restrito. Software que atenda a um grupo ou organização será usado por pessoas com

habilidades, papéis e preferências variadas. Portanto, o projeto do software sofrerá impacto de fatores sociais, motivacionais, econômicos e até políticos que não aparecem em software destinado a um único indivíduo.*

O estudo formal de atividades grupais é tarefa geralmente a cargo de pesquisadores das áreas administrativa ou econômica. As técnicas utilizadas, derivadas da psicologia social e a antropologia, incluem a pesquisa contextual (onde o usuário é entrevistado no seu contexto de trabalho), a observação participativa (na qual o pesquisador participa temporariamente das atividades do grupo sendo estudado) e o projeto participativo (onde os usuários formam uma parceria com os projetistas do software).

---

* Este é provavelmente o principal motivo do fracasso, até o momento, da "automação de escritórios" que visa utilizar o computador como único meio de comunicação escrita entre os indivíduos, eliminando totalmente a circulação de informações impressas.

CAPÍTULO 2

CARACTERÍSTICAS

DAS INTERFACES

DE USUÁRIO

Este Capítulo descreve algumas características necessárias à criação de boas interfaces de usuário. Essas características parecerão óbvias para quem estiver familiarizado com interfaces modernas. No entanto, muitas vezes elas são deixadas de lado em função da pressão que normalmente existe para completar as aplicações o mais rapidamente possível (levando ao projeto de interfaces fáceis de implementar ao invés de fáceis de usar). Na maioria das vezes o projeto da interface não é formalizado, o que permite a introdução involuntária de inconsistências após a fase de projeto por parte dos programadores responsáveis pela implementação.

Apesar das razões para um projeto de interface de baixa qualidade serem compreensíveis, as interfaces dos aplicativos resultantes podem ser de qualidade deplorável. Para projetar uma interface é necessário um projeto completo, bem documentado, e elaborado a partir de um conjunto claro de premissas básicas, como por exemplo as abaixo detalhadas. A sua descrição é baseada em [FOLEY 82] e [MYERS 89].

2.1. Realimentação

A realimentação é um componente essencial do diálogo: sem ela, haveria apenas um monólogo frustrante para o usuário. Mas diferentemente de um diálogo entre dois seres humanos, o diálogo homem-máquina só dará realimentação se esta tiver sido planejada e programada.

O tipo mais simples de realimentação é o eco dos caracteres digitados na tela do monitor. Outra forma de realimentação é a mudança de cor de uma opção de menu ao ser escolhida, indicando ao usuário que a sua escolha foi registrada.

A forma mais útil de realimentação é aquela que indica ao usuário que a operação solicitada já foi completada. Normalmente isto é indicado através de uma tela nova ou modificada que indica os resultados explicitamente, ou pela emissão de um som característico.

Outro tipo de realimentação, necessária apenas se o tempo necessário para completar as ações solicitadas for superior a um tempo mínimo (por exemplo, meio segundo), é uma indicação de que o computador está processando a solicitação do usuário (como por exemplo, um indicador do percentual do trabalho já efetuado). Algumas vezes esta indicação pode ser substituída pela apresentação gradual dos resultados parciais já computados.

No caso excepcional das ações necessárias consumirem uma quantidade muito grande de tempo, ou se houver necessidade de agrupar vários comandos antes que as ações correspondentes sejam executadas, pode-se exibir uma descrição precisa daquilo que a aplicação considere que tenha sido solicitado pelo usuário. Este tipo de realimentação é valioso para usuários novatos, principalmente se existir a possibilidade de haver ambigüidade ou dependências do contexto atual. Uma das aplicações dessa realimentação ocorreu com o advento de interfaces em linguagem natural. Por

exemplo, o sistema **Intellect** [IBM 83], que aceita consultas a bancos de dados em linguagem natural, exhibe após cada frase de consulta uma tradução para comandos de uma linguagem computacional de consulta ("query language").

Finalmente, é importante que a posição na tela em que a realimentação é apresentada seja fácil de ser detectada pelo usuário. Existe uma tendência a usar áreas fixas na tela para a exibição da realimentação e das mensagens de erro, mas isto cria uma descontinuidade visual na tela. É melhor exhibir a realimentação na posição da tela em que o usuário estiver trabalhando, por exemplo modificando o formato do cursor ou apresentando uma pequena janela ligada graficamente ao ponto do erro. Alternativamente, pode-se usar realimentação sonora.

2.2. Disponibilidade de ajuda para o aprendizado

A maioria das interfaces de usuário precisa ser projetada para atender a usuários com variados níveis de experiência. *Usuários novatos* são aqueles que estão começando a aprender os conceitos básicos e o funcionamento da aplicação; *usuários intermediários* são aqueles que já aprenderam a realizar algum trabalho produtivo com a aplicação; e *usuários experientes* são aqueles que dominam e conhecem (quase) toda a aplicação e seus recursos. Note que esta classificação em três níveis de experiência é apenas uma aproximação: o nível de experiência dos usuários é na realidade um valor contínuo.

A primeira necessidade dos usuários novatos é provavelmente uma demonstração guiada (tutorial), com exemplos, dos comandos básicos da aplicação, onde o usuário tenha que apenas escolher entre as alternativas apresentadas.

Existem outras duas maneiras de ajudar os usuários: através da exibição de *mensagens de prontidão** e com um *sistema de ajuda***. Diferentemente da realimentação, que indica ao usuário a aceitação de suas ações, as mensagens de prontidão têm como objetivo sugerir ao usuário o que ele deve fazer a seguir. Quanto mais experiente for o usuário, portanto, menor será a quantidade ou tamanho apropriado dessas mensagens, principalmente se a sua exibição influir na velocidade da interação do usuário com a aplicação. Várias interfaces possuem vários níveis dessas mensagens, permitindo ao usuário escolher o nível desejado.

Mensagens de prontidão podem assumir várias formas. A forma mais direta é uma mensagem textual indicando explicitamente o próximo passo (por exemplo, "Digite a data: "). Alternativamente, um sintetizador de voz poderia instruir o usuário. No entanto, existem formas mais sutis, como por exemplo a exibição com cores diferenciadas dos elementos que podem ser selecionados, ou a mudança de formato do cursor na medida em que é posicionado sobre partes diferentes da tela para indicar, por meio de ícones, as ações possíveis (por exemplo, um cursor com formato de escala poderia indicar a necessidade de se entrar com um valor numérico). Estes tipos de avisos de prontidão geralmente não são apropriados para usuários novatos.

Na maioria das interfaces modernas, o diálogo é iniciado pelo computador através da apresentação de um menu. Este tipo de interação é apropriado para usuários novatos, porque eles são guiados através das opções disponíveis. Menus são apropriados para usuários intermediários apenas se eles podem ser apresentados com velocidade suficiente para que o usuário não tenha que esperar o término da sua exibição na tela. Já para usuários experientes é conveniente a existência de um mecanismo alternativo de acesso aos comandos da aplicação, como por exemplo o uso de *teclas*

---

* "Prompt messages", em inglês.

** "Help engine", em inglês.

aceleradoras, também denominadas *teclas de atalho** (por exemplo, pode-se associar comandos freqüentemente usados a "teclas de função").

Um sistema de ajuda permite ao usuário obter informação adicional sobre os vários comandos disponíveis e sobre as técnicas de interação. Tipicamente o usuário aciona um comando chamado "Help" ou "Ajuda", que deve poder ser ativado em qualquer momento da interação entre o usuário e a aplicação, sempre através do mesmo mecanismo. Ao retornar do sistema de ajuda, a aplicação deve ficar exatamente no mesmo estado em que estava antes do usuário acionar esse sistema.

Os sistemas de ajuda são geralmente organizados em vários níveis. Imediatamente após seu acionamento, o sistema de ajuda deve fornecer ao usuário *ajuda sensível ao contexto*, isto é, informação a respeito do comando ou subdiálogo da aplicação que ele estiver utilizando no momento. A partir desse primeiro nível de ajuda, o usuário deve poder solicitar ajuda a respeito de outros assuntos relacionados, efetuar pesquisas entre os tópicos disponíveis e ter acesso a um índice de tópicos de ajuda disponíveis.

A existência de um sistema de ajuda é apropriada mesmo quando forem exibidos menus e mensagens de prontidão, porque o usuário terá uma oportunidade a mais de receber informação mais detalhada do que aquela que pode ser indicada através de uma mensagem de prontidão. Mesmo usuários experientes podem ocasionalmente esquecer algum detalhe, principalmente em aplicações grandes e complexas. Neste caso, o sistema de ajuda é uma maneira rápida e simples de se chegar à informação necessária, sem precisar interromper o trabalho com a aplicação para consultar a sua documentação impressa (por exemplo, manuais).

---

* "Accelerator keys" ou "shortcut keys", respectivamente, em inglês.

2.3. Erros do usuário e reversão

Todo usuário, independentemente de seu nível de experiência, comete erros durante a utilização das aplicações. É necessário que exista um mecanismo que permita a reversão desses erros, isto é, a não interrupção da sessão, sob pena do usuário cometer outros ainda mais graves. A reversão de erros visa diminuir o nível de frustração do usuário, que levaria a uma diminuição da produtividade do usuário.

Quase todas as interfaces interativas interpretam a tecla de retrocesso* para apagar o último caractere digitado (quando deve ser dada uma seqüência de caracteres pelo usuário). Outra forma de reversão de erros semelhante seria a capacidade de cancelar toda a seqüência de caracteres já digitada.

Uma outra forma de reversão de erros seria a possibilidade do usuário cancelar a execução de um comando enquanto ele é executado. A aplicação deve ter seu estado restaurado para o estado *anterior* ao início da execução do comando.

Ainda melhor seria a possibilidade de reverter** (ou desfazer) o efeito do último comando acionado pelo usuário, mesmo após o encerramento da sua execução e independentemente dos efeitos do comando (como por exemplo, alterações em arquivos). Esta forma de reversão de erros é também a mais custosa: a aplicação deve continuamente armazenar seu estado, ou mudanças de estado, e fornecer um mecanismo para recriar um estado anterior. No entanto, ela libera o usuário para explorar os recursos da

---

* "Backspace", em inglês.

** "Undo", em inglês.

aplicação que ele não conhece, já que a qualquer efeito indesejado de um comando, este pode ser revertido imediatamente.

Algumas interfaces fornecem a possibilidade de cancelar o efeito de certos comandos solicitando ao usuário explicitamente que os confirme. No entanto, esta estratégia aumenta o número total de ações do usuário necessárias para desempenhar seu trabalho. Uma alternativa para esse caso é exigir uma ação explícita do usuário apenas se ele quiser cancelar o comando: a escolha do próximo comando pelo usuário pode servir como confirmação para o anterior.

Uma aplicação que forneça a possibilidade de cancelar comandos logicamente exige um esforço extra de programação, principalmente para aqueles comandos que resultam em alterações substanciais nos valores armazenados nas estruturas de dados utilizadas pela aplicação. Uma alternativa seria exigir do usuário confirmação apenas no caso dos comandos que, se executados, demandariam um esforço significativo para serem revertidos (como por exemplo um comando que apaga um arquivo).

2.4. Tempo de resposta

O tempo de resposta de uma aplicação é crítico para a viabilidade do uso de uma aplicação, embora nem sempre o tempo de resposta esteja sob total controle da aplicação (por exemplo, no caso de sistemas operacionais multitarefa).

Quando uma aplicação tem tempos de resposta perceptíveis pelo usuário, surgem dois custos: o primeiro e imediato é o tempo dispendido pelo usuário aguardando a resposta da aplicação; o segundo é o tempo que o usuário precisa para reorganizar as suas idéias e poder continuar com a sua tarefa, uma vez que a aplicação tiver respondido.

Mesmo quando a aplicação pode controlar por completo o tempo de resposta, não existe uma medida simples do tempo de resposta ideal: o tempo de resposta deve ser proporcional às expectativas do usuário. Quanto maior ou mais significativa for a tarefa em execução, maior será a tolerância do usuário a tempos de resposta maiores.

No entanto, existem alguns casos onde há limites muito rígidos para o tempo de resposta. Por exemplo, no caso de ações reflexas (não pensadas) do usuário, como a digitação de um caractere ou a movimentação do cursor, a resposta tem que ser aparentemente imediata, o que impõe um tempo de resposta máximo de alguns poucos décimos de segundo.

Outro caso com limites rígidos para o tempo de resposta ocorre nas ações simples, onde o usuário não espera que ocorra uma demora significativa. Se ocorrer, o usuário é levado a pensar no motivo da demora da resposta, perdendo assim a sequência de raciocínio que estava seguindo em sua interação com a aplicação. Tradicionalmente, nos ambientes de mainframe, um tempo máximo de dois segundos era considerado satisfatório. No entanto, com a disseminação de computadores pessoais e estações de trabalho cada vez mais poderosos, a maioria dos usuários espera uma resposta quase instantânea (poucos décimos de segundo), exceto no caso de solicitações envolvendo grandes volumes de cálculos ou pesquisas em disco.

2.5. Consistência

Uma interface consistente deve ter um modelo conceitual, uma sintaxe e um visual uniformes, sem exceções ou condições especiais.

Alguns exemplos simples de consistência na saída da aplicação são a utilização uniforme de cores (a mesma cor codifica sempre o mesmo tipo de informação), o posicionamento lógico das mensagens sempre no mesmo lugar (embora o local físico possa não ser fixo, como por exemplo se for usada a

posição do cursor), ou a colocação das opções nos menus sempre na mesma ordem (o que permite ao usuário usar a sua memória visual e motora para selecionar as opções).

No caso dos dados de entrada das aplicações podemos citar como exemplos de consistência o fato das teclas especiais (como mudança de linha, retrocesso* e outras) terem sempre o mesmo significado e poderem ser usadas sempre que houver entrada de texto, ou a disponibilidade permanente de comandos globais para efetuar o acesso ao sistema de ajuda ou para cancelar o comando corrente.

A consistência das entradas leva a estruturas mais simples, não só minimizando a quantidade de ações necessárias por parte do usuário (o que tem um impacto direto na sua produtividade), mas também facilitando o processo de aprendizado e diminuindo a quantidade de erros.

A consistência deve ainda permitir ao usuário deduzir que se uma operação pode ser aplicada a um certo tipo de objeto, também possa ser aplicada a objetos similares. Por exemplo, se os campos de entrada de texto possuem um valor padrão inicial que pode ser recuperado mesmo depois que o usuário tenha digitado outro valor, o mesmo recurso deve estar disponível para outros tipos de entrada de valores, como por exemplo a escolha de um elemento de uma lista.

Um dos objetivos da consistência deve ser o de permitir ao usuário *generalizar* seu conhecimento a respeito do comportamento da aplicação em uma determinada situação para outras situações. Outro objetivo é evitar a frustração do usuário que ocorre quando a aplicação não tem um comportamento lógico e compreensível.

---

* "Carriage return" e "Backspace" respectivamente, em inglês.

2.6. Visual estruturado

Freqüentemente é necessário que a aplicação exiba grandes quantidades de informação para o usuário. Se ela for apresentada de maneira não estruturada, ela pode ser difícil de ser percebida e entendida. Uma maneira de estruturar a informação é criar áreas distintas na tela, quase sempre chamadas de janelas*, para a exibição de cada tipo de informação (como por exemplo mensagens de erro, o status da aplicação, menus e a representação dos dados da aplicação), facilitando a localização da informação pelo usuário. As interfaces mais recentes, como a do Macintosh (Apple), o Windows (Microsoft) e os sistemas baseados no X-Windows (como o Motif, da Open Software Foundation, e o OpenWindows da Sun, incorporado no Solaris), permitem inclusive ao usuário rearranjar a posição e tamanho das várias janelas na tela. Embora as áreas possam se sobrepor, as suas bordas as separam claramente.

A percepção da estrutura visual pelo usuário pode ser reforçada através da utilização de cores, formas geométricas, tipos de linhas e intensidades. Embora a cor seja a forma mais fácil de distinção entre elementos, é importante lembrar que uma parte significativa dos usuários sofre de problemas de daltonismo. A redundância da codificação permite que cada usuário focalize aquela diferença que é mais facilmente reconhecível para ele.

Uma maneira de criar um visual estruturado é a utilização da exibição seletiva. Informações gráficas podem ser exibidas em níveis de zoom diferentes. Para qualquer tipo de informação é possível exibir apenas uma parte dos dados. Neste caso é necessário que exista um recurso que permita

---

* "Windows", em inglês.

ao usuário saber a posição e tamanho relativos dos dados exibidos em relação ao total (como por exemplo, barras de deslocamento^{*}).

2.7. Memorização mínima

O processo de aprender a usar uma aplicação exige a memorização de certos tipos de informação. É importante, no entanto, que a memorização necessária seja o mais simples possível, quando não for possível eliminá-la por completo. Uma maneira de atingir este objetivo é através da consistência (veja a seção 2.5), mas existem outras maneiras. Por exemplo, não é desejável que os parâmetros necessários à execução de um determinado comando tenham que ser dados pelo usuário em uma única ordem prefixada.

Muitas vezes, os projetos das interfaces exigem memorização desnecessária por parte do usuário, como por exemplo quando o usuário é obrigado a se referir aos objetos sendo manipulados por número ao invés de usar seu nome.

A utilização de nomes ao invés de números vale não apenas para casos extremos (nenhum sistema operacional interativo exige do usuário que use apenas números para referir-se aos arquivos), mas também em casos mais sutis, como o caso das interfaces de janelas nas quais o usuário deve selecionar a janela a ser usada pelo seu número! Esta situação ocorre no ambiente interativo do compilador Borland C++ e no depurador de programas CodeView Versão 4.0 da Microsoft. Nas duas situações o usuário é privado da utilização de nomes mnemônicos, dificultando não só a memorização, mas também o aprendizado.

---

* "Scroll Bars", em inglês.

Uma maneira adicional de diminuir a memorização necessária é a utilização de processos de seleção onde todos os objetos sejam exibidos explicitamente (ou representados por um ícone). Desta maneira o usuário precisa apenas aprender a reconhecer os objetos ao invés de memorizar a sua existência.

Algumas interfaces exigem memorização no caso do usuário utilizar o sistema de ajuda. Quando o acesso ao sistema de ajuda modifica a tela de maneira tão significativa que o usuário não pode mais ver a situação que o levou a pedir ajuda, ele precisa ter memorizado essa situação.

CAPÍTULO 3

FORMALIZAÇÃO DAS INTERFACES DE USUÁRIO

Todo sistema para o desenvolvimento de interfaces de usuário utiliza algum *modelo* de interface de usuário. Este modelo serve de base à notação usada pelos UIDSs* para descrever as interfaces de usuário. Ainda, o modelo escolhido influencia fortemente a implementação do próprio UIDS.

Existe uma variedade de técnicas para descrever interfaces de usuário. Estas técnicas podem ser divididas em duas categorias, dependendo de sua utilização no projeto ou na implementação das interfaces de usuário. As notações de projeto são geralmente informais, já que seu objetivo principal é registrar as idéias do projetista. Por outro lado, a notação usada para a implementação de uma interface de usuário tem que ser formal para viabilizar a implementação. Neste Capítulo descreveremos notações úteis para a implementação. Várias partes desta descrição são baseadas em [GREEN 86].

A notação usada por um UIDS define a variedade de interfaces de usuário que podem ser produzidas com esse UIDS. Embora nem sempre seja um objetivo desejável, a generalidade de um UIDS depende da capacidade da notação em descrever a maior variedade possível de interfaces.

---

* Abreviatura de "User Interface Development Systems".

A capacidade de uma notação pode ser avaliada de duas maneiras. A primeira medida é o *poder de expressão*, isto é, o conjunto das interfaces de usuário que podem ser descritas através dessa notação. Quanto maior for este conjunto, tanto mais poderosa será a notação. Para determinar o poder de expressão de uma notação, o problema deve ser convertido em um problema equivalente na teoria formal de linguagens. Na maioria dos casos é possível encontrar respostas precisas.

A segunda maneira de avaliar o poder de expressão de uma notação é a sua *capacidade útil*. A capacidade útil é o conjunto de interfaces de usuário que podem ser descritas facilmente com a notação em questão. Naturalmente, a capacidade útil sempre é um subconjunto da capacidade de descrição. Como a maioria dos projetistas de interfaces tendem a usar interfaces fáceis de descrever na notação escolhida, existe uma influência da notação sobre a qualidade das interfaces a serem criadas. Não existe nenhuma medida objetiva para a capacidade útil. Como consequência, neste Capítulo usamos a capacidade de descrição.

Um dos modelos de interface de usuário mais conhecidos é o modelo de Seeheim (descrito em [PFAFF 85]). Este modelo é bastante geral e possui uma descrição independente de um UIDS específico, o que facilita o estudo do modelo por si mesmo.

O modelo de Seeheim divide a interface de usuário em três componentes (veja a Figura 3.1). O *componente de apresentação* lida com a representação física da interface de usuário. Isto inclui os dispositivos de entrada e saída, as técnicas de interação e de exibição. Este componente é a única parte da interface que se comunica diretamente com os dispositivos de entrada e saída. Os outros componentes da interface de usuário não podem comunicar-se com os dispositivos de entrada e saída. O componente de apresentação pode ser considerado como o nível léxico da interface de usuário. O componente *controle do diálogo* é responsável pela estrutura do diálogo entre o usuário e a aplicação. Ele pode ser considerado como o nível sintático da interface de usuário. Finalmente, o componente *interface da*

aplicação é responsável pela chamada dos processos da aplicação. Este componente pode ser considerado como sendo o nível semântico da interface de usuário.

De um ponto de vista lógico, os três componentes podem ser considerados processos separados, que se comunicam através da passagem de *tokens*, similares aos tokens usados na construção de compiladores (veja, por exemplo, [ASU 86]). Um token consiste de um nome, que identifica o seu tipo, associado a uma coleção de valores. Um *token de entrada* é aquele que vai do usuário para a aplicação, um *token de saída* vai da aplicação para o usuário.

No modelo de Seeheim a aplicação pode modificar o estado da interface de usuário enviando um token de saída para ela. Assim a interface de usuário pode considerar a aplicação como mais um dispositivo de entrada e reagir a ela da mesma maneira que reage ao usuário. Outra vantagem é que a interface de usuário não precisa chamar rotinas da aplicação para determinar seu próprio estado.

A maioria das pesquisas já efetuadas sobre UIDSs foi direcionada ao componente de controle do diálogo. Conseqüentemente, existe muito mais conhecimento sobre este componente do que sobre os outros dois.

3.1. Introdução aos modelos de diálogo

Existem basicamente três tipos de modelos de controle de diálogo: redes de transição, gramáticas livres de contexto e eventos.

Uma técnica mais recente de modelagem usa estadogramas* (definidos em [HAREL 87]). Estes não serão abordados nesta resenha, pois não são mencionados na literatura consultada como ferramenta de modelagem para sistemas de interface, e devido à extensão que nosso trabalho já assumiu.

Os três modelos de controle de diálogo usados na literatura consultada são apresentados informalmente a seguir. As suas definições formais serão apresentadas na seção 3.2.

---

* "State charts", em inglês.

Figura 3.2 - Uma rede de transição simples

3.1.1. Redes de Transição

O modelo de *redes de transição** (abreviado RT) é baseado em diagramas de transição, basicamente um grafo orientado onde a cada vértice é associado um estado e a cada aresta é associado um evento. Os estados representam os estados do diálogo entre o usuário e o computador (um estado é considerado *inicial* e existe um conjunto de estados *finais*). A cada aresta é associado um evento (um token de entrada). O diálogo passa do estado A para o estado B se existir uma aresta entre esses estados rotulada pela ação efetuada pelo usuário.

Um exemplo simples de RT é o da Figura 3.2. O componente de apresentação produz o token "botão" quando o usuário apertar ou soltar o botão do mouse e "mover" toda vez que o usuário mover o mouse.

Esta forma de RT descreve a sequência de eventos gerados pelo usuário, mas não indica as respostas geradas pelo computador. Uma maneira de descrever as respostas é associando ações aos estados, ou às transições.

---

* Do original em inglês "transition networks".

As ações devem ser executadas quando o estado for atingido ou a transição ocorrer.

Este tipo de RTs tem dois problemas. Serve apenas para descrever uma variedade limitada de diálogos, e as descrições tendem a ser muito grandes.

O segundo problema pode ser resolvido particionando a rede, dividindo-a em uma RT principal e um conjunto de subredes que servem como arestas de outras (sub)redes. A utilização de subredes permite dividir a

descrição do diálogo em um conjunto de unidades lógicas; por exemplo, a cada subrede poderia corresponder um comando da interface de usuário.

O uso de subredes de transição não aumenta o poder de expressão das RTs, já que cada uso de uma subrede pode ser substituído pelos estados e arestas da subrede. Mas se for permitida a definição de subredes utilizando a si próprias recursivamente, então o poder de expressão aumenta. Este tipo de RT é denominado de *rede de transição recursiva*, abreviado RTR.* Um exemplo de RTR, descrevendo o desenho de uma linha poligonal, é exibido na Figura 3.3.

As *redes de transição aumentadas*, abreviadas RTA**, são uma extensão das RTRs, definidas pela primeira vez em [WOODS 70]. Além das redes e subredes, uma RTA possui um conjunto de registradores e um conjunto de funções. Os registradores podem conter valores arbitrários e são visíveis apenas dentro do componente de controle do diálogo (a aplicação não tem acesso a eles). As funções podem efetuar qualquer cálculo com os valores dos registradores e atribuir novos valores aos registradores, mas não podem ter acesso aos valores da aplicação. As funções são associadas às arestas da RTA e são executadas quando a aresta estiver para ser utilizada. Caso a função retorne um valor "verdadeiro", a aresta poderá ser usada; caso contrário, é como se a aresta não existisse momentaneamente.

O poder de expressão das RTAs é ainda maior do que o das RTRs, já que é possível incluir dependência do contexto utilizando os registradores e as funções. Vários pesquisadores propõem a utilização de RTAs como a base do componente de controle de diálogo de UIDSs. O autômato a pilha interativo sugerido em [OLSEN 84] também se assemelha a uma RTA.

---

* "Recursive Transition Network", abreviado RTN, em inglês.

** "Augmented Transition Network", abreviado ATN, em inglês.

```
<linha> ::= botão <ponto_final>  
<ponto_final> ::= mover <ponto_final> | botão
```

Figura 3.4 - Exemplo de gramática livre de contexto

Um dos principais problemas com o modelo de RTs é o tratamento de ações inesperadas do usuário. Quando o diagrama de transições está em um determinado estado e a ação do usuário não corresponde a nenhum dos rótulos das arestas saindo desse estado, o diálogo não pode continuar.

Existem várias soluções para esse problema. A solução mais simples consiste em ignorar a ação do usuário, mas não é muito satisfatória, já que o usuário não é informado que a sua ação está sendo ignorada e nem recebe informação sobre como atingir um novo estado. Outra solução consiste em usar um rótulo genérico "outros eventos" que conduza a um estado de tratamento de erros.

A principal vantagem do modelo de RTs é a sua natural representação gráfica. Os diagramas de transições podem ser exibidos e editados facilmente em terminais gráficos.

3.1.2. Gramáticas Livres de Contexto

O modelo de *gramáticas livres de contexto* (abreviado GLC) é baseado na descrição do diálogo usuário-computador através do uso de gramáticas livres de contexto. A motivação é a analogia com o diálogo entre seres humanos, que utiliza uma linguagem natural cuja estrutura é descrita por uma gramática.

No entanto, existe uma diferença importante entre o diálogo entre seres humanos e o diálogo usuário-computador: enquanto no primeiro é utilizada apenas uma linguagem, no segundo o usuário e o computador utilizam

```
<poligonal> ::= botão <repetição>  
<repetição> ::= <comando> <repetição> | <comando>  
<comando>    ::= fim | mover <comando> | botão <continuar>  
<continuar>  ::= retrocesso | <comando> <próximo>  
<próximo>    ::= mover | retrocesso
```

Figura 3.5 - A poligonal descrita por uma gramática

linguagens diferentes. A existência dessas duas linguagens cria diversos problemas ao utilizar GLCs para descrever interfaces de usuário.

Na prática, é usada apenas uma gramática para descrever a linguagem utilizada pelo usuário para se comunicar com o computador; as respostas do computador são descritas de alguma outra maneira. Os tokens de entrada servem como símbolos terminais da gramática. Veja por exemplo a Figura 3.4, que descreve o mesmo diálogo que a Figura 3.2. Outro exemplo é o da Figura 3.5, que descreve o mesmo diálogo que a Figura 3.3.

O exemplo da Figura 3.4 não descreve as respostas do computador. Para incluí-las na descrição pode-se adotar uma estratégia semelhante à usada com as RTs: as respostas do computador podem ser associadas às produções da gramática. Essas respostas devem ser executadas quando a produção for usada na análise sintática da entrada do usuário.

Um dos principais problemas da associação de ações da aplicação às produções da gramática decorre do fato de que o momento em que a produção é usada na análise sintática da entrada depende do tipo de algoritmo (de análise sintática) utilizado. No caso dos algoritmos "top-down", a produção começa a ser utilizada quando o usuário entrar com o primeiro símbolo terminal que aparece do lado direito da produção. No entanto, no caso dos algoritmos "bottom-up" a produção só é utilizada após o reconhecimento de todos os símbolos terminais que aparecem no lado direito da produção.

3.1.3. Eventos

O modelo de *eventos* popularizou-se mais recentemente do que os dois modelos anteriores. Este modelo é baseado no conceito de eventos de entrada utilizado em alguns softwares gráficos. Nestes softwares os dispositivos de entrada são considerados fornecedores de eventos. Cada dispositivo de entrada gera um ou mais eventos quando o usuário interage com o dispositivo. Cada evento tem um nome ou número que identifica a natureza da interação, mais vários valores extras que caracterizam a interação. Por exemplo, no caso de um mouse, um evento "mover" é gerado toda vez que o mouse for movimentado pelo usuário. Os dados associados ao evento são os valores correntes das coordenadas x e y do cursor. Um evento separado é gerado também a cada vez que um dos botões do mouse é apertado ou solto.

Os eventos são colocados em uma fila quando são gerados, e a aplicação retira os eventos da fila um por vez chamando uma das rotinas do software gráfico. Em alguns casos a aplicação pode especificar o tipo de evento que precisa, o que dá à aplicação algum controle sobre os eventos que são gerados. Nesses softwares gráficos há um número fixo e predefinido de eventos, e os eventos podem ser gerados apenas pelos dispositivos de entrada.

O *modelo de eventos* é uma extensão das idéias acima. No modelo de eventos há uma quantidade arbitrária de eventos. Alguns desses eventos são gerados pelos dispositivos de entrada, mas outros eventos são gerados internamente ao componente de controle do diálogo. O programador tem a liberdade de definir novos tipos de eventos mais apropriados para uma determinada aplicação.

Quando um evento for gerado, ele é enviado para um ou mais *tratadores de eventos*. Um tratador de eventos é um processo (definido por um módulo ou rotina) que é capaz de processar certos tipos de eventos. Quando um tratador de eventos recebe um evento que é capaz de processar, ele executa um procedimento (semelhante a um método nas linguagens

orientadas a objeto, veja por exemplo [GOLDBERG 83]). Este procedimento pode efetuar alguns cálculos, gerar novos eventos, chamar rotinas da aplicação, criar novos tratadores de eventos ou destruir tratadores de eventos já existentes.

O comportamento de um tratador de eventos é definido por um molde*. Um molde é composto de várias seções que definem os argumentos do tratador de eventos, os eventos que é capaz de processar, suas variáveis locais e os procedimentos utilizados para processar esses eventos.

Quando um tratador de eventos é criado, seu molde deve ser especificado conjuntamente com valores para seus argumentos. O resultado do processo de criação do tratador de eventos é um nome único utilizado para referenciar o tratador de eventos. Vários tratadores de eventos podem ser criados a partir de um mesmo molde. Cada um dos tratadores de eventos criados a partir de um mesmo molde pode ter um estado local diferente (parâmetros e variáveis locais).

Após a criação de um tratador de eventos, ele permanece ativo até que seja destruído, ou por decisão própria ou por outro tratador de eventos. Somente os tratadores de eventos ativos podem responder a eventos.

No modelo de eventos, uma interface de usuário é descrita por um conjunto de moldes que serão usados para definir os tratadores de eventos da interface. No início da execução é criada uma instância de um desses tratadores de eventos para servir como tratador de eventos principal da interface de usuário. Este tratador será responsável pela criação (possivelmente indireta) de todos os demais tratadores de eventos que compõem a interface. Conceitualmente, todos os tratadores de evento ativos da interface de usuário são executados de forma concorrente, processando os eventos à medida que eles chegam. Como cada tratador de eventos somente pode processar um evento por vez, cada tratador de eventos pode ser

---

* "Template", em inglês.

considerado um processo monitor (veja [NAKA 91] para um uso semelhante de monitores associados a objetos).

Existem duas possíveis origens para os eventos no componente de controle do diálogo. A primeira origem são os outros dois componentes da interface de usuário, já que tanto o componente de apresentação como o componente de interface da aplicação enviam tokens ao componente de controle do diálogo. Estes tokens são convertidos em eventos, que são então enviados aos tratadores de eventos que deverão processá-los. Um tratador de eventos pode declarar os tokens que deseja receber convertidos. Portanto, mais de um tratador de eventos pode processar o mesmo token.

A segunda origem dos eventos é o próprio componente de controle do diálogo. Um tratador de eventos pode enviar um evento para qualquer um dos outros tratadores de eventos ativos. Neste caso, o nome do tratador de eventos que receberá o evento é especificado explicitamente na operação de envio do evento. Desta forma os eventos servem como protocolo de comunicação entre os tratadores de eventos.

A Figura 3.6 exhibe um exemplo de molde de tratador de eventos que descreve o diálogo para o desenho de uma poligonal sem possibilidade de retrocesso. A definição do molde do tratador de eventos é composta de quatro seções:

- a primeira seção declara quais tokens o tratador de eventos será capaz de processar e o mapeamento deles (`tbotão` e `tmover`) nos procedimentos de tratamento de evento (`BOTÃO` e `MOVER`);
- a seção `VAR` declara as variáveis locais ao tratador de eventos;
- a terceira seção especifica o processamento a ser efetuado para cada evento recebido; e
- a última seção contém os comandos a serem executados quando o tratador de eventos for instanciado.

```
EVENT_HANDLER linha;  
  
TOKEN  
    tbotão    BOTÃO;  
    tmover    MOVER;  
  
VAR  
    inteiro ESTADO;  
    ponto    PRIMEIRO, ULTIMO;  
  
EVENT BOTÃO DO BEGIN  
    IF ESTADO = 0 THEN  
        PRIMEIRO := posição corrente;  
        ESTADO   := 1;  
    ELSE BEGIN  
        ULTIMO   := posição corrente;  
        destroy(self);  
    END;  
END;  
  
EVENT MOVER DO BEGIN  
    IF ESTADO = 1 THEN  
        exibalinha(PRIMEIRO, posição corrente);  
    END;  
END;  
  
INIT  
    ESTADO := 0;  
  
END EVENT_HANDLER linha;
```

Figura 3.6 - Um exemplo de tratador de eventos

Note que neste exemplo, para manter a sua simplicidade, as respostas da aplicação estão incluídas no processamento da entrada do usuário.

```
EVENT HANDLER poligonal_completa;

TOKEN
    tbotão      BOTÃO;
    tmover      MOVER;
    tretrocesso RETROCESSO;
    tcancelar   CANCELAR;
    tfim        FIM;

VAR
    contador: inteiro;
    lista    : lista de pontos;
    estado   : inteiro;

EVENT BOTÃO DO BEGIN
    IF estado = 0 THEN
        lista      := posição atual;
        estado     := 1;
        contador   := 1;
    ELSE
        adicione posição atual à lista;
        contador := contador + 1;
    END;
END;

EVENT MOVER DO BEGIN
    IF estado = 1 THEN
        desenhe linha da posição anterior até a atual
    END;
END;

EVENT RETROCESSO DO BEGIN
    IF contador > 1 THEN
        apague último ponto da lista;
        contador := contador - 1;
    END;
END;
```

Figura 3.7 - Um molde de tratador de eventos
para a poligonal completa

```
ELSE
    saída("Primeiro ponto não pode ser apagado");
END;
END;

EVENT CANCELAR DO BEGIN
    devolva lista vazia;
    destroy(self);
END;

EVENT FIM DO BEGIN
    devolva lista;
    destroy(self);
END;

INIT
    estado := 0;

END EVENT HANDLER poligonal_completa;
```

Figura 3.7 - Um molde de tratador de eventos
para a poligonal completa (continuação)

A Figura 3.7 exhibe o molde do tratador de eventos equivalente à RTR da Figura 3.3 e à GLC da Figura 3.5.

Uma das principais vantagens do modelo de eventos é a sua capacidade de descrever diálogos compostos por vários diálogos executados simultaneamente*. Neste tipo de diálogos o usuário pode estar participando

---

* Este tipo de diálogos é denominado, em inglês, de "multithreaded dialog". O conceito de *thread* tem origem nos sistemas operacionais: um thread é um caminho seguido pela execução de instruções dentro de um programa. Alguns sistemas operacionais modernos, como por exemplo Mach, OS/2 e Windows NT permitem a execução de vários threads dentro de uma aplicação simultaneamente. Este recurso é também chamado de *execução multitarefa intra-aplicação*.

de vários diálogos, com comunicação entre eles ou não, ao mesmo tempo. O usuário tem a liberdade de mudar de um diálogo para outro a qualquer momento durante a sua interação com a aplicação. Este tipo de diálogo ocorre frequentemente em aplicações baseadas em janelas. Como cada tratador de eventos pode ser considerado como um processo separado, resulta bastante simples de descrever diálogos compostos por diálogos simultâneos com o modelo de eventos.

Como exemplo de um diálogo composto por vários diálogos simultâneos, considere um editor de textos que permite ao usuário editar vários arquivos simultaneamente. Cada um dos arquivos é exibido em uma janela separada. É possível criar uma coleção de moldes de tratadores de eventos que descrevam as interações que ocorrem em uma janela de edição. Quando o usuário edita um arquivo, são criadas instâncias desses tratadores de eventos. Conseqüentemente, cada janela de edição possui os seus próprios tratadores de eventos.

A edição dentro de uma janela poderia ser descrita adequadamente utilizando qualquer um dos três modelos de diálogo, mas os modelos de rede de transição e gramáticas livres de contexto são insuficientes para descrever a comunicação entre as janelas. Considere, por exemplo, um recurso da interface do tipo recortar-e-colar*, que poderia ser parte do editor de textos multiarquivo. O usuário pode recortar ou copiar um trecho de texto de um arquivo para outro, o que envolve a comunicação entre diálogos executando em duas janelas. Enquanto no modelo de eventos esta situação pode ser descrita facilmente enviando eventos de uma janela para a outra, os outros dois modelos precisariam ser estendidos para poder descrever essa situação.

---

* "Cut-and-Paste", em inglês.

3.2. Definição formal dos modelos de diálogo

Nesta seção apresentaremos definições formais dos três modelos de diálogo. O objetivo desta formalização é fornecer uma base para a comparação dos três modelos e um meio para deduzir as suas propriedades.

Enfatizaremos nestas definições o mecanismo utilizado para descrever as ações do usuário. Assumiremos como hipótese que o programa aplicativo não pode modificar diretamente o estado da interface de usuário (para modificar o estado da interface de usuário o aplicativo deve enviar um token de saída para a interface de usuário). A principal consequência desta hipótese é que a interface de usuário não pode usar procedimentos da aplicação para estender a própria interface. No entanto, se este tipo de comportamento fosse permitido, acabaríamos perdendo a noção de um módulo de interface de usuário separado do restante do código da aplicação.

3.2.1. Redes de Transição

Iniciaremos o desenvolvimento da definição formal do modelo de redes de transição com as redes de transição simples (compostas por apenas um diagrama, abreviadas por RTS daqui em diante), e as estenderemos depois para incluir as redes de transição recursivas.

No caso das RTSs, seu comportamento pode ser descrito por um autômato de estados finitos modificado. Os estados deste autômato correspondem aos estados no diagrama de transição e as arestas são convertidas em transições de estado. O alfabeto de entrada é o conjunto dos tokens que podem ser gerados pelo componente de apresentação. Assim uma RTS pode ser descrita por uma 7-upla $M = (Q, \Sigma, P, \delta, \gamma, q_0, f)$, onde

1. Q é um conjunto finito de estados, correspondente aos estados no diagrama;
2. Σ é um conjunto finito de símbolos de entrada, que são os tokens de entrada gerados pelo componente de apresentação;
3. P é um conjunto finito de ações, que são as ações usadas como rótulos dos estados do diagrama de transição;
4. δ é uma função de $Q \times \Sigma$ em Q , chamada de função de transição de estados;
5. γ é uma função de $Q \times \Sigma$ em P , chamada de função de ação;
6. $q_0 \in Q$ é o estado inicial de M ;
7. $f \subset Q$ é o conjunto de estados finais de M .

O comportamento de uma RTS é ligeiramente diferente daquele de um autômato de estados finitos. A configuração de uma RTS é representada pelo estado corrente q do autômato. Quando a RTS recebe um símbolo de entrada a , representando uma ação do usuário, ela muda para o estado determinado pela função de transição de estados. Ao efetuar esta transição de estados, a RTS emite o nome da ação: o novo estado q' é o valor de $\delta(q, a)$ e a ação p emitida é $\gamma(q, a)$.

Ao iniciar-se o diálogo, a RTS está no estado inicial q_0 . Na medida em que o usuário gera tokens de entrada, a RTS mudará de estado em estado, seguindo as arestas rotuladas pelos tokens de entrada. O fim do diálogo ocorre quando a RTS entra em um dos estados finais do conjunto f . A RTS reconhece as sequências de tokens de entrada que a forçam a mudar de seu estado inicial até um de seus estados finais.

No caso das RTRs (redes de transição recursivas) é necessário um mecanismo mais potente. Este mecanismo é o autômato determinístico a pilha (ADP) [AHO 72]. Usaremos uma versão modificada do ADP, denominada

uma rede de transição (RT), na nossa definição das redes de transição recursivas. A principal diferença entre uma RTS e uma RT é o uso de subdiagramas nas RTs. Nas redes de transição recursivas, o controle é transferido para um subdiagrama quando é encontrada uma aresta rotulada por ele. Quando o percurso do subdiagrama tiver sido completado, o controle é transferido de volta para o nó no final da aresta que ativou o subdiagrama. Como os subdiagramas podem ser ativados recursivamente, é necessário fornecer algum mecanismo para armazenar os nós no final das arestas que ativam subdiagramas. Este é o principal motivo para usar um ADP como base para a definição.

Uma RT é uma 9-upla $M = (Q, \Sigma, P, \Gamma, \delta, \gamma, q_0, Z_0, f)$, onde

1. Q é um conjunto finito de estados, correspondente aos estados no diagrama;
2. Σ é um conjunto finito de símbolos de entrada, que são exatamente os tokens de entrada gerados pelo componente de apresentação;
3. P é um conjunto finito de ações, que são as ações usadas como rótulos dos estados do diagrama de transição;
4. Γ é um conjunto finito de símbolos de pilha;
5. δ é uma função de $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ em $Q \times \Gamma^*$, chamada de função de transição de estados, onde ϵ indica que o símbolo de entrada deve ser ignorado e $*$ indica a operação de fechamento transitivo;
6. γ é uma função de $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ em P , chamada de função de ação;
7. $q_0 \in Q$ é o estado inicial de M ;
8. $Z_0 \in \Gamma$ é o símbolo inicial na pilha;
9. $f \subset Q$ é o conjunto de estados finais de M .

A configuração de uma RT é um par ordenado $(q, z) \in Q \times \Gamma^*$, onde q é o estado corrente da RT e z é o conteúdo da sua pilha. Toda vez que a RT recebe um token de entrada do componente de apresentação, ela muda para uma nova configuração e emite o nome de uma ação. Se a RT tiver a configuração $(q, \alpha Z)$ com $\alpha \in \Gamma$ e $Z \in \Gamma^*$ e receber o símbolo de entrada a , então o novo estado e o símbolo no topo da pilha são determinados por $\delta(q, a, \alpha)$. A ação emitida é determinada por $\gamma(q, a, \alpha)$.

No início do diálogo, a configuração da RT é (q_0, Z_0) . Na medida em que os tokens de entrada são recebidos, a RT muda de uma configuração para outra de acordo com a função de transição de estados. A RT reconhece as sequências de tokens de entrada que a forçam a mudar da sua configuração inicial para uma das configurações (q, Z_0) , onde $q \in f$.

3.2.2. Gramáticas Livres de Contexto

O modelo de gramáticas livres de contexto é baseado nas gramáticas livres de contexto, estendidas para incluir a chamada das ações da aplicação. Um modelo de gramática livre de contexto é uma 5-upla $G = (N, T, R, P, S)$, onde

1. N é um conjunto finito de símbolos chamados não-terminais;
2. T é um conjunto finito de símbolos chamados terminais, e há um símbolo em T para cada token de entrada produzido pelo componente de apresentação;
3. R é um conjunto finito de símbolos que correspondem às ações associadas às produções;
4. P é um conjunto finito de produções da forma

$$n \rightarrow \alpha, r$$

onde $n \in N$, $\alpha \in (N \cup T)^*$ e $r \in R$.

5. $S \in N$ é o símbolo inicial da gramática.

Uma produção pode ser considerada como uma regra que determina que o não-terminal do lado esquerdo pode ser substituído pelos símbolos do lado direito. Por exemplo, se tivermos a produção

$$X \rightarrow abc$$

e a seqüência de símbolos

$$\alpha X \beta$$

então podemos substituir o não-terminal X pela seqüência de símbolos abc , resultando na seqüência de símbolos

$$\alpha abc \beta$$

Esta operação é denominada de um passo de derivação e é denotada por:

$$\alpha X \beta \rightarrow \alpha abc \beta$$

Uma seqüência de passos de derivação tal como

$$\alpha_0 \rightarrow \alpha_1 \rightarrow \dots \rightarrow \alpha_n$$

pode ser abreviada por

$$\alpha_0 \rightarrow^* \alpha_n$$

A linguagem, $L(G)$, descrita por uma gramática G , é definida da seguinte maneira:

$$L(G) = \{ x \in T^* \mid S \rightarrow^* x, \text{ onde } S \text{ é o símbolo inicial de } G \}.$$

Se G descreve o componente de controle de diálogo de uma interface de usuário, $L(G)$ contém todas as seqüências corretas de ações do usuário (e somente estas). Uma gramática livre de contexto G pode ser usada para

produzir um analisador sintático que reconheça as sequências de ações do usuário em $L(G)$.

Os algoritmos de análise sintática podem ser divididos basicamente em dois grupos: "bottom-up" e "top-down". Um algoritmo de análise sintática "top-down" começa com o símbolo inicial da gramática e o expande usando as produções. Na medida em que os passos de derivação são efetuados, os terminais do lado direito das produções são comparados com os tokens gerados pelo componente de apresentação. A análise sintática continua enquanto os terminais coincidirem com os tokens de entrada. Maiores detalhes sobre análise sintática "top-down" podem ser encontrados em livros-texto sobre compiladores, como [ASU 86] e [SETZER 83].

Quando uma produção é expandida numa análise sintática "top-down", a ação associada à produção é emitida. Uma produção é selecionada para ser expandida tão logo o primeiro terminal que pode ser gerado pelo seu lado direito coincida com o token corrente de entrada. Conseqüentemente, a ação associada à produção é executada quando for reconhecido o início de um comando.

Um analisador sintático "bottom-up" funciona na direção oposta: ele acumula tokens de entrada até que se componha, com os últimos símbolos lidos e/ou reconhecidos, o lado direito de uma produção. Nesse momento, esses símbolos são substituídos pelo não-terminal do lado esquerdo da produção, e o processo continua. Este processamento de substituições continua até que o símbolo inicial da gramática seja atingido. Mais detalhes sobre analisadores sintáticos "bottom-up" podem ser encontrados também em [ASU 86].

Nos algoritmos de análise sintática "bottom-up" a produção não pode ser usada antes de que tenham sido reconhecidos todos os símbolos do seu lado direito, portanto a ação executará apenas quando o comando representado pela produção tenha sido digitado por inteiro.

Das considerações anteriores conclui-se que o momento exato em que a ação é emitida depende do algoritmo de análise sintática utilizado. Este é um dos principais problemas no uso de gramáticas como a notação para o componente de controle de diálogo de uma interface de usuário. A descrição do diálogo dependerá do algoritmo de análise sintática utilizado. Em geral, não é nada fácil portar uma descrição de um diálogo baseada em gramática de um UIDS para outro.

3.2.3. Eventos

Os dois componentes principais do modelo de eventos são os eventos e os tratadores de eventos. Na prática, os tratadores de eventos são embutidos em linguagens de programação de propósito geral. Embora isto dê ao modelo de eventos uma potência considerável, tende a obscurecer as suas propriedades gerais. Para caracterizar este modelo de diálogo, construiremos uma versão da sua imersão numa linguagem hospedeira abstrata, possuindo uma estrutura bastante simples e de interpretação evidente.

Em um sistema abstrato de eventos, um evento e é uma tripla $e = (i, m, d) \in E$ (o conjunto, possivelmente infinito, de todos os eventos), onde

1. $i \in I$, onde I é um conjunto finito de símbolos usados como nomes para os tratadores de eventos;
2. $m \in M$, onde M é um conjunto finito de símbolos usados como nomes de eventos;
3. $d \in D$, onde D é um domínio (possivelmente infinito) usado para valores dos eventos. Neste Capítulo, D geralmente é o conjunto dos inteiros.

A linguagem hospedeira abstrata, já exemplificada nas Figuras 3.6 e 3.7, consistirá aqui de apenas um número finito de registradores e seis comandos simples.

Cada tratador de eventos possui um número finito de registradores, cada um capaz de armazenar um valor de D ou de I , e trata determinados *tipos de eventos*. Para cada tipo de evento tratado, o tratador de eventos possui um registrador. Os demais registradores podem ser usados pelo projetista da interface de usuário para armazenar valores temporários computados durante o processamento de um evento, ou para armazenar um valor usado pelo tratador de eventos no processamento de vários eventos.

Cada tipo de evento é processado por um procedimento separado, chamado de *procedimento de tratamento de evento*. O corpo deste procedimento consiste de uma seqüência de comandos. Seis tipos de comandos, que são as primitivas do modelo de eventos, podem aparecer dentro de um procedimento de tratamento de evento. Os seis tipos de comandos são os seguintes (r indica um certo registrador):

1. $r := \langle \text{expressão} \rangle$

Este comando calcula um novo valor para um dos registradores do tratador de eventos. A expressão é composta de registradores, constantes e operadores. A expressão não pode referenciar o conteúdo de registradores de outros tratadores de eventos ou valores de variáveis da aplicação.

2. $i \leftarrow m \ r$

Este comando cria um novo evento de nome m e valor r , e o envia para o tratador de eventos i , que pode ser especificado ou através de uma constante ou de um valor de registrador.

3. $r := \text{create}(t, v_1, v_2, \dots, v_n)$

Este comando cria um novo tratador de eventos baseado no molde t de tratador de eventos passado como parâmetro. O comportamento deste novo

tratador de eventos é definido pelo molde. Os demais parâmetros $v_1, v_2, \dots, v_n$ são os valores iniciais (símbolos de I ou de D) para os registradores (pela ordem, $r_1, r_2, \dots, r_n$) do novo tratador de eventos. Se não houver valores suficientes para inicializar todos os registradores, os registradores não especificados recebem o valor zero.

4. `destroy(r)`

Este comando destrói o tratador de eventos identificado pelo nome (símbolo de I) armazenado em r .

5. `IF <condição> THEN <comando> ELSE <comando> END`

O comando `IF` permite a seleção da execução entre duas sequências de comandos, dependendo do valor da condição. A condição pode consistir de registradores, constantes, operadores aritméticos (+, -, etc) e relacionais (<, >, etc) e conectivas lógicas. Os comandos `IF` podem ser encaixados.

6. `call proc(arg1, arg2, ..., argn)`

Este comando é a chamada de um dos procedimentos próprios da aplicação, programado na linguagem hospedeira. Os argumentos do procedimento são expressões compostas de registradores, constantes e operadores. O procedimento da aplicação não pode retornar um valor para o tratador de eventos.

Os seis tipos de comandos acima são suficientes para o nosso objetivo. Vale a pena destacar o que não pode ser feito com esses comandos: não há como produzir uma repetição dentro de um procedimento de tratamento de um evento, nem pode ser chamado um procedimento de tratamento de evento de dentro de um procedimento de tratamento de evento. Isto elimina a possibilidade de construir uma repetição infinita dentro de um procedimento de tratamento de evento. Como consequência, cada procedimento de tratamento de evento terminará sua execução dentro de um tempo finito. Isto

Um tratador de eventos TE é uma 5-upla $TE = (j, k, Q, R, P)$, onde

1. j é a quantidade de tipos de eventos processados pelo tratador de eventos;
2. k é a quantidade de registradores disponíveis no tratador de eventos, com a condição que $j \leq k$;
3. $Q \in E^*$ é a fila de eventos^{**} do tratador de eventos. Os eventos em Q já foram enviados a este TE , mas ainda não foram processados;
4. $R \subseteq (D \cup D)^k$ é o conjunto de símbolos dos registradores de TE ;
5. P é um conjunto de pares ordenados, cada um deles formado por um tipo de evento $m \in M$ e seu correspondente procedimento de tratamento de evento.

Uma *configuração* de um tratador de eventos é um par ordenado, formado pelo conteúdo da sua fila de eventos e de seus registradores. Esta configuração é representada como (q, ρ) , onde $q \in E^*$ e $\rho \in R$.

Um sistema de eventos SE é um conjunto finito de tratadores de eventos. A quantidade de tratadores de eventos em um sistema de eventos pode variar dinamicamente, na medida em que forem criados novos tratadores de eventos e destruídos os antigos.

O comportamento de um sistema de eventos obedece às seguintes regras:

1. Quando um token de entrada é recebido do componente de apresentação, ele é convertido pelo sistema de controle do diálogo em um evento, que é adicionado ao fim das filas de eventos dos tratadores de eventos que declararam interesse naquele token.

---

^{**} "Event queue", em inglês.

2. O processamento de um evento é uma operação atômica: em outras palavras, a configuração de um tratador de eventos não reflete a execução individual dos comandos contidos nos procedimentos de tratamento de evento. Se a configuração corrente do tratador de eventos for (eq, ρ) , a próxima configuração do tratador de eventos será (q', ρ') . Nesta nova configuração ρ' é o conjunto dos valores dos registradores após a execução do procedimento de tratamento de evento correspondente ao evento e , e $q' \in E^*$, sendo que q' pode ser diferente de q (se o procedimento de tratamento de evento enviar novos eventos para o mesmo tratador de eventos). A execução de um procedimento de tratamento de evento também pode adicionar eventos no fim das filas de eventos pertencentes a outros tratadores de eventos do sistema.
3. Esta regra lida com a execução concorrente de vários tratadores de eventos. Pela regra anterior, cada processamento de um evento é uma operação atômica: cada tratador de eventos pode processar apenas um evento por vez. Os eventos enviados a um tratador de eventos em particular são processados na ordem em que são enviados. Não existe nenhuma restrição sobre a ordenação temporal dos eventos enviados aos vários tratadores de eventos, para permitir que o componente de controle do diálogo seja distribuído em vários processadores.

A maneira normal de implementar o modelo de eventos é o uso de um preprocessador que converte os programas escritos na linguagem de eventos em programas escritos numa linguagem de programação normal. Os procedimentos de tratamento de evento já são escritos na linguagem de programação que é alvo do preprocessador. Conseqüentemente, o preprocessador somente precisa traduzir os componentes da linguagem que são específicos da linguagem de eventos.

Para algumas linguagens de programação pode ser necessária uma biblioteca de tempo de execução* para tratar as partes da linguagem de eventos que não são facilmente implementadas na linguagem de programação alvo. Exemplos deste tipo de recursos são o escalonamento da execução dos procedimentos de tratamento de evento, o envio dos eventos e a criação de múltiplas instâncias do mesmo tratador de eventos. Maiores detalhes sobre a implementação de linguagens de eventos usando esta estratégia podem ser encontrados em [CHIA 85] e [GREEN 85].

3.3. Poder de expressão

Analisaremos agora o poder de expressão dos três modelos: redes de transição recursivas (RTRs), gramáticas livres de contexto (GLCs) e eventos. Usaremos dois enfoques para esta comparação: o primeiro é puramente teórico, baseado na teoria formal de linguagens, enquanto o segundo é mais prático, mostrando como dois dos modelos podem ser convertidos no terceiro.

Uma RTR é essencialmente um autômato determinístico a pilha (ADP), pois trata-se de um autômato de estados finitos com transições que podem ser autômatos. A única diferença entre uma RTR e um ADP é a função γ , que associa ações às transições da RTR. No entanto, esta função não tem nenhum efeito sobre as linguagens aceitas pela RTR. Portanto, podemos concluir que o poder de expressão de uma RTR é equivalente ao de um ADP.

Em relação às GLCs, deve ser observado que na prática apenas um subconjunto das linguagens geradas por essas gramáticas é usado na construção de interfaces de usuário. A necessidade de utilizar um algoritmo

---

* "Run-time system", em inglês.

de análise sintática eficiente limita o uso às linguagens livres de contexto determinísticas.

É fato conhecido da teoria formal de linguagens que o conjunto das linguagens livres de contexto determinísticas são aquelas reconhecidas por ADPs (veja, por exemplo, [AHO 72] ou [LOMET 73]). No entanto, **não é verdade** que o poder de expressão das RTRs e das GLCs sejam iguais.

Para verificar a afirmação anterior, considere uma RTR com dois caminhos iniciando no estado inicial, tal que os primeiros $k - 1$ tokens nestes dois caminhos sejam iguais, mas os k -ésimos tokens dos dois caminhos sejam diferentes. Para este tipo de diagrama de transições é possível construir um ADP que trabalhe com um "lookahead" de k tokens de entrada para determinar corretamente o caminho a seguir. Isto é essencialmente o que um analisador sintático do tipo LR(k) faria para determinar o caminho a seguir.

Desafortunadamente, essa estratégia não pode ser adotada no caso de interfaces de usuário, já que pode haver ações associadas aos estados percorridos antes de se obter o k -ésimo token. Estas ações não poderiam ser usadas quando o usuário gerasse os correspondentes tokens de entrada (já que não saberíamos sobre qual dos dois caminhos estamos), privando o usuário, portanto, de toda e qualquer realimentação sintática e semântica.

Na prática é utilizada uma restrição sobre as RTRs, de tal maneira que em cada estado seja possível decidir qual o caminho correto a seguir conhecendo-se apenas o próximo token de entrada. Com esta restrição, as RTRs resultam ser equivalentes às gramáticas com "lookahead" de um token, que são um subconjunto das GLCs determinísticas.

Em relação ao modelo de eventos, observamos que na prática ele é embutido em uma linguagem de programação. Portanto, o poder de expressão do modelo de eventos corresponde ao poder de expressão de uma máquina de Turing, o que lhe dá um poder de expressão maior do que o dos outros dois modelos.

Uma outra maneira de comparar o poder de expressão dos três modelos passa pela utilização dos algoritmos de conversão entre eles apresentados nas seções seguintes. Esses algoritmos permitem converter qualquer RTR e qualquer GLC para o modelo baseado em eventos, o que caracteriza o fato de que o poder de expressão do modelo de eventos é no mínimo igual ao dos outros dois.

Para mostrar que o modelo de eventos possui um poder de expressão maior, é necessário exibir uma interface de usuário que possa ser descrita no modelo de eventos, mas que não possa ser descrita usando os outros dois modelos. Para tanto, basta voltar a considerar o editor de múltiplos arquivos, com o recurso de recortar e copiar trechos de texto de uma janela para outra, descrito na página 41.

Nesse exemplo, cada arquivo tem a sua própria janela e um conjunto de tratadores de eventos que implementam as funções básicas de edição dentro daquela janela. Como cada janela possui seus próprios tratadores de eventos, o usuário pode mudar de uma janela para outra a qualquer momento. O recurso de recortar-e-colar permite ao usuário mover (ou copiar) texto de um lugar para outro dentro de uma janela ou de uma janela para outra. Este recurso pode ser ativado a qualquer momento, independentemente dos estados dos diálogos nas janelas envolvidas.

Para mover um trecho de texto, o usuário seleciona o texto a ser transferido, escolhe o comando cortar ou copiar de um menu, indica a nova posição para o texto e escolhe o comando colar do menu. A posição em que o texto é inserido não precisa ser selecionada imediatamente após a escolha do comando cortar ou copiar do menu.

Um tratador de eventos para o recurso de recortar e colar é exibido na Figura 3.8. Este tratador de eventos processa cinco tipo de tokens. Os tokens **cortar**, **copiar** e **colar** são gerados quando o usuário seleciona o comando correspondente no menu. O token **selecionar** é gerado toda vez que o usuário seleciona um trecho de texto dentro de uma das janelas de edição. Este token identifica tanto o texto selecionado como a janela na qual foi

efetuada a seleção. O token `posição` é gerado toda vez que o usuário escolhe uma posição. Este token inclui a janela e a posição (dentro dessa janela) selecionados pelo usuário. A notação `evento.valor` indica o uso de um valor associado ao evento.

Como o recurso de recortar-e-colar pode ser ativado a qualquer instante durante a sessão de edição e sua execução pode misturar-se com outras operações de edição, este tipo de diálogo não pode ser descrito por redes de transição ou gramáticas. Outros tipos de diálogos difíceis ou impossíveis de descrever usando redes de transição e gramáticas são descritos em [HILL 86].

```
EVENT HANDLER recortar_e_colar;
```

```
TOKEN
```

```
    cortar      CORTAR;  
    copiar      COPIAR;  
    colar       COLAR;  
    selecionar  SELECIONAR;  
    posição     POSIÇÃO;
```

```
VAR
```

```
    seleção:      texto;  
    dono:         inteiro;  
    seleção_atual: texto;  
    dono_seleção: inteiro;  
    posição_atual: ponto;  
    dono_posição: inteiro;
```

Figura 3.8 - Um tratador de eventos para recortar-e-colar

```
EVENT SELECIONAR DO BEGIN
    seleção := SELECIONAR.seleção;
    dono     := SELECIONAR.dono;
END;

EVENT POSIÇÃO DO BEGIN
    posição_atual := POSIÇÃO.posição;
    dono_posição  := POSIÇÃO.dono;
END;

EVENT CORTAR DO BEGIN
    dono ← "apagar_seleção"
    seleção_atual := seleção;
    dono_seleção  := dono;
END;

EVENT COPIAR DO BEGIN
    seleção_atual := seleção;
    dono_seleção  := dono;
END;

EVENTO COLAR DO BEGIN
    dono_posição ← "adicioneseleção" posição e seleção atual
END;

END EVENT HANDLER recortar_e_colar;
```

Figura 3.8 - Um tratador de eventos
para recortar-e-colar (continuação)

3.3.1. Conversão de Redes de Transição em Sistemas de Eventos

A conversão de um conjunto de diagramas de transição recursiva em um sistema de eventos é baseada na construção de um molde de tratador de

eventos para cada diagrama. Um tratador de eventos será criado a partir deste molde toda vez que o diagrama precisa ser percorrido e destruído quando o percurso do diagrama terminar. Ao ser criado, cada tratador de eventos receberá dois parâmetros: a instância que o chamou e o estado seguinte a ser atingido nessa mesma instância. Ao atingir um estado final, o tratador de eventos envia um evento "continuar" à instância que o chamou, tendo como valor o estado em que esse diagrama deve continuar seu processamento.

O primeiro passo no algoritmo de conversão é o cálculo do valor da função LEADING para cada um dos subdiagramas da rede de transição. Dado um subdiagrama d , seja $L(d)$ o conjunto das sequências de tokens em Σ que são reconhecidas por d . Cada elemento de $L(d)$ rotula um caminho do estado inicial até um dos estados finais de d . A função LEADING é definida como

$$\text{LEADING}(d) = \{ a \in \Sigma \mid \exists S \in \Sigma^* \text{ e } aS \in L(d) \}$$

A função LEADING é o conjunto de primeiros símbolos de entrada que o subdiagrama está esperando quando é chamado. O procedimento para o cálculo de LEADING é similar ao utilizado para calcular a relação FIRST para gramáticas livres de contexto (veja [ASU 86]), já que pode existir uma transição contendo um diagrama a partir de um estado inicial.

Nos passos restantes, os diagramas são considerados um por vez, sendo construído um molde de tratador de evento para cada diagrama. O diagrama sendo considerado pelo algoritmo será chamado de diagrama corrente.

No segundo passo do algoritmo construiremos um esqueleto do molde de tratador de eventos para o diagrama corrente. Este esqueleto contém o molde completo, exceto os procedimentos de tratamento de evento.

Para construir o esqueleto, é necessário determinar o conjunto de tokens do diagrama corrente. Este conjunto pode ser construído começando com o conjunto vazio e adicionando ao conjunto cada token de entrada que é usado como rótulo de alguma aresta do diagrama. Se o rótulo da aresta for um nome de diagrama, adicione o conjunto LEADING desse diagrama ao

```
EVENT HANDLER nome_diagrama(chamador: I; destino: inteiro);  
  
TOKEN  
    token1 evento1;  
    token2 evento2;  
    ...  
    tokenn eventon;  
  
VAR  
    ativo: inteiro;  
    estado: inteiro;  
    temp: inteiro;  
  
...  
  
INIT  
    ativo := 1;  
    estado := número do estado inicial do diagrama;  
  
END EVENT HANDLER nome_diagrama;
```

Figura 3.9 - Esqueleto para molde de tratador de eventos

conjunto de tokens. Cada token deste conjunto de tokens é usado para criar uma linha na seção TOKEN do molde do tratador de eventos, já que são estes os tokens que o tratador de eventos estará interessado em processar.

Usaremos os diagramas de transição do exemplo da poligonal (veja a Figura 3.3, à página 31) para ilustrar o algoritmo de conversão. Para esses diagramas, os conjuntos de tokens são

```
conjunto_token(principal) = { botão, mover, fim }  
conjunto_token(comando)   = { mover, retrocesso, botão, fim }
```

A estrutura do esqueleto de um tratador de eventos é exibido na Figura 3.9. O molde de tratador de evento tem o mesmo nome que o diagrama correspondente. Os dois argumentos do tratador de evento são: o nome do

tratador de evento que o ativou, e o número do estado no fim da aresta corrente no diagrama correspondente ao tratador de evento que o ativou.

A variável local **ativo** indica se o tratador de eventos está processando entrada no momento ou não: se o valor da variável for 1, os tokens de entrada são processados; caso contrário, eles são ignorados.

A variável local **estado** registra o estado corrente do diagrama. Ela é inicializada com o estado inicial do diagrama de transição correspondente. A terceira variável local é uma variável temporária utilizada para armazenar o valor retornado pelo operador **create**.

O terceiro passo do algoritmo é a construção dos procedimentos de tratamento de evento para os eventos correspondentes aos tokens contidos no conjunto de tokens. Cada um dos procedimentos de tratamento de evento tem a seguinte estrutura básica:

```
EVENT eventoi DO BEGIN
  IF ativo = 1 THEN
    IF estado = n1 THEN
      ...
    ELSE IF estado = n2 THEN
      ...
    ELSE IF estado = nm THEN
      ...
    END
  END
END
END
```

Há um comando **IF** no procedimento para cada aresta rotulada pelo token correspondente. Neste caso o procedimento atualiza a variável **estado** para o estado no fim da aresta e executa quaisquer ações associadas a esse estado. Também há um comando **IF** para cada aresta rotulada por um subdiagrama que contém o token correspondente em seu conjunto **LEADING**. Neste caso, o procedimento ativa o subdiagrama e passa o evento para ele. A

```
EVENT BOTÃO DO BEGIN
  IF ativo = 1 THEN
    IF estado = 1 THEN
      estado := 2;
      registra primeiro ponto;
    END;
  END;
END;

EVENTO MOVER DO BEGIN
  IF ativo = 1 THEN
    IF estado = 2 THEN
      ativo := 0;
      temp := create(Comando, esta_instância, 2);
      temp ← "Mover" Mover;
    END;
  END;
END;

EVENT Fim DO BEGIN
  IF ativo = 1 THEN
    IF estado = 2 THEN
      ativo := 0;
      temp := create(Comando, esta_instância, 2);
      temp ← "Fim" Fim;
    END;
  END;
END;
```

Figura 3.10 - Procedimentos de tratamento de eventos para o diagrama Poligonal

aplicação deste processo ao diagrama Poligonal resulta nos procedimentos de tratamento de evento exibidos na Figura 3.10.

```
EVENT continuar DO BEGIN
    ativo := 1;
    estado := continuar;
END;
```

Figura 3.11 - O tratador do evento Continuar do diagrama Poligonal

O quarto e último passo no algoritmo de conversão é a construção do procedimento de tratamento de evento para o evento *continuar*. Este evento sinaliza o fim do percurso de um subdiagrama que foi ativado pelo diagrama corrente. O valor deste evento é o estado no fim da aresta que ativou o subdiagrama. O tratador do evento *continuar* tem a seguinte estrutura:

```
EVENT continuar DO BEGIN
    ativo := 1;
    estado := continuar;
    IF estado = estado1 THEN
        chamar procedimento para estado1;
    ELSE IF estado = estado2 THEN
        chamar procedimento para estado2;
    ...
    ELSE IF estado = estadon THEN
        chamar procedimento para estadon;
    END;
END;
```

Há um comando *IF* neste procedimento para cada estado que aparece no final de uma aresta rotulada pelo subdiagrama. Este comando *IF* chama o procedimento que rotula esses estados.

O procedimento correspondente ao evento *continuar* do diagrama *Poligonal* é exibido na Figura 3.11. Depois que cada diagrama tenha sido considerado, a conversão estará completa.

3.3.2. Conversão de Gramáticas Livres de Contexto em Sistemas de Eventos

Uma maneira de converter as gramáticas livres de contexto em tratadores de eventos é usar os tratadores de eventos para simular um dos algoritmos de análise sintática. O algoritmo de conversão descrito nesta seção é baseado no algoritmo de análise sintática para gramáticas LL(1), que é um algoritmo "top-down" (veja [ASU 86] ou [SETZER 83]).

A escolha de LL(1) deve-se a dois fatos: em primeiro lugar, pelo menos um UIDS bem conhecido é baseado nessa técnica (**Syngraph**, veja [OLSEN 83]), e em segundo, o algoritmo de análise sintática para LL(1) é bastante simples e intuitivo, ao mesmo tempo que abrange uma variedade bastante grande de gramáticas. Os algoritmos de análise sintática tipo LR(1) são mais potentes, mas são consideravelmente mais complicados. O uso de um algoritmo de análise sintática mais complicado não tem nenhuma vantagem neste contexto.

A gramática da Figura 3.5, na página 34, descreve a mesma interface de usuário que o conjunto de diagramas de transição de estado exibidos na Figura 3.3. Usaremos esta gramática para ilustrar a conversão de gramáticas livres de contexto em tratadores de eventos.

O algoritmo de conversão é baseado na construção de um molde de tratador de evento para cada não-terminal da gramática. Uma instância deste tratador de evento é criada a cada vez que o não-terminal correspondente precisa ser expandido na análise sintática. Em geral, a produção para um não-terminal terá a forma

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n \mid \epsilon \quad (1)$$

Os símbolos $\alpha_1, \alpha_2, \dots, \alpha_n$ representam sequências não vazias de terminais e não-terminais, enquanto o símbolo ϵ representa a sequência vazia. Esta produção estabelece que o não-terminal A pode ser substituído por $\alpha_1, \alpha_2, \dots$,

```

FIRST(<poligonal>) = { botão }
FIRST(<repetição>) = { fim, mover, botão }
FIRST(<comando>)   = { fim, mover, botão }
FIRST(<continuar>) = { retrocesso, fim, mover, botão }
FIRST(<próximo>)   = { mover, retrocesso }

FOLLOW(<comando>)  = { fim, mover, botão, retrocesso }
FOLLOW(<continuar>) = { fim, mover, botão, retrocesso }
FOLLOW(<próximo>)  = { fim, mover, botão, retrocesso }

```

Figura 3.12 - FIRST e FOLLOW para a gramática da Figura 3.5

ou α_n , ou pela string vazia. Todos os não-terminais têm uma produção desta forma, exceto que a alternativa vazia pode não existir.

O algoritmo de conversão consiste dos seguintes passos. Primeiro, é necessário calcular os conjuntos FIRST e FOLLOW. Estes conjuntos têm as seguintes definições:

$$\text{FIRST}(X) = \{ a \mid a \in T, X \rightarrow^* aR, R \in (N \cup T)^* \}$$

$$\text{FOLLOW}(X) = \{ a \mid a \in T, S \rightarrow^* \alpha X a \beta, \alpha, \beta \in (N \cup T)^* \}$$

O conjunto $\text{FIRST}(X)$ contém os terminais que podem aparecer no início de uma sequência de símbolos derivada do não-terminal X . Note a similaridade entre FIRST e a função LEADING para as redes de transição recursivas.

O conjunto $\text{FOLLOW}(X)$ contém os terminais que podem vir depois de X em alguma derivação a partir do símbolo inicial da gramática.

A Figura 3.12 exhibe as relações FIRST e FOLLOW para a interface de usuário do exemplo. Veja [ASU 86] para uma apresentação de algoritmos para calcular FIRST e FOLLOW.

Os passos restantes do algoritmo de conversão consideram os não-terminais um por vez e constroem o molde de tratador de eventos para

ele. O não-terminal sendo considerado momentaneamente pelo algoritmo é denominado não-terminal corrente.

O segundo passo consiste na construção de um esqueleto para o molde de tratador de eventos. Para construir este esqueleto é necessário computar o conjunto de tokens para o tratador de eventos. Este conjunto contém todos os tokens em que o tratador de eventos estará interessado. Assumindo que a produção seja da forma da equação (1) acima, o conjunto de tokens pode ser computado da seguinte maneira: cada símbolo de cada alternativa deve ser examinado. Se o símbolo for terminal, ele é adicionado ao conjunto de tokens da produção. Se o símbolo é não-terminal, então todos os símbolos do seu conjunto FIRST são adicionados ao conjunto de tokens. Se a produção tiver uma alternativa ϵ , então todos os símbolos do conjunto FOLLOW do não-terminal do lado esquerdo da produção são adicionados ao conjunto de tokens.

O formato do esqueleto para o molde do tratador de eventos é essencialmente o mesmo que foi usado para as redes de transição recursivas (veja a Figura 3.9). O nome do molde do tratador de eventos é o mesmo do nome do não-terminal que representa. Os dois parâmetros do tratador de eventos são o nome do tratador de eventos que o ativou e o estado corrente do tratador de eventos que o ativou.

O terceiro passo do algoritmo é a construção dos procedimentos de tratamento de evento. Primeiro, numere os símbolos do lado direito da produção: o primeiro símbolo da primeira alternativa recebe o número 1, o segundo símbolo o número 2, e assim por diante, até que todos os símbolos tenham sido numerados (exceto a alternativa ϵ , que não é numerada). Estes números, chamados de estados da produção, são usados para controlar a posição corrente dentro da análise sintática. A seguir, os procedimentos de tratamento de evento são construídos, um por vez. Haverá um procedimento para cada token no conjunto de tokens. Esses procedimentos têm essencialmente o mesmo formato que aqueles usados para as redes de transição recursivas.

```
EVENT Mover DO BEGIN
  IF ativo = 1 THEN
    IF estado = 0 THEN
      ativo := 0;
      temp := create(comando, esta_instância, 2);
      temp ← "Mover" Mover;
    END;
    IF estado = 3 THEN
      ativo := 0;
      temp := create(comando, esta_instância, 3);
      temp ← "Mover" Mover;
    END;
  END;
END;

EVENT Retrocesso DO BEGIN
  IF ativo = 1 THEN
    IF estado = 0 THEN
      chamador ← "Continuar" estado_corrente;
      destroy(esta_instância);
    END;
    IF estado = 3 THEN
      ativo := 0;
      temp := create(próximo, esta_instância, 3);
      temp ← "Retrocesso" Retrocesso;
    END;
  END;
END;
```

Figura 3.13 - Alguns dos procedimentos de tratamento de evento para a produção <comando>

As ações executadas pelo procedimento de tratamento de evento dependem do estado corrente da produção. Se o símbolo no estado corrente é um terminal, o estado avança para o próximo símbolo na produção ou retorna para a produção que chamou a corrente (se for atingido o fim da produção).

Se o símbolo é não-terminal, então deve ser ativado o tratador de eventos correspondente a esse não-terminal. A aplicação deste processo à produção <comando> resulta, entre outros, nos procedimentos de tratamento de evento da Figura 3.13.

O quarto passo do algoritmo de conversão é a construção do procedimento de tratamento do evento *continuar*. Este evento é recebido quando um tratador de evento ativado pelo tratador de evento corrente tiver reconhecido seu não-terminal.

O processamento do evento *continuar* depende do não-terminal correspondente ser ou não o último símbolo da alternativa. O valor do evento é o número de estado do não-terminal que ativou o tratador de eventos. A forma básica do procedimento de tratamento do evento *continuar* é:

```
EVENT Continuar DO BEGIN
  estado := continuar + 1;
  ativo  := 1;
  IF continuar =  $n_a$  THEN
    chamar procedimento para alternativa a;
    chamador ← "continuar" estado_corrente;
    destroy(esta_instância);
  END;
  ...
  IF continuar =  $n_z$  THEN
    chamar procedimento para alternativa z;
    chamador ← "continuar" estado_corrente;
    destroy(esta_instância);
  END;
END;
```

No procedimento acima, n_a até n_z são os números de estado dos não-terminais que aparecem no fim das alternativas.

Depois que cada não-terminal tiver sido processado por estes passos, a conversão está pronta.

3.3.3. Análise dos Algoritmos de Conversão

Existem dois significados para a palavra "eficiente" no contexto dos algoritmos de conversão das duas seções anteriores. O primeiro significado é que os algoritmos podem converter eficientemente a descrição da interface de usuário de uma forma para a outra. O segundo significado é que o tratador de eventos produzido pelo algoritmo de conversão é uma implementação eficiente do diálogo. Ambos significados são importantes no contexto dos UIDS e são analisados a seguir.

A eficiência dos algoritmos de conversão propriamente ditos pode ser analisada assim: os cálculos de LEADING, FIRST e FOLLOW são $O(N^2)$ no pior caso, onde N é a quantidade de diagramas de transição ou a quantidade de alternativas nos lados direitos das produções da gramática. O pior caso é raramente atingido na prática. Os demais passos dos algoritmos são repetidos M vezes, onde M é a quantidade de diagramas de transição ou a quantidade de produções da gramática. Cada uma dessas iterações implica em um número reduzido de percursos da estrutura de dados representando cada diagrama de transição ou produção. Portanto, esta parte dos algoritmos é $O(M)$. Conseqüentemente, a eficiência destes algoritmos é quadrática no pior caso, e na média eles são quase lineares.

A eficiência da implementação deve ser comparada com as técnicas normalmente utilizadas para implementar redes de transição recursivas e gramáticas livres de contexto. Uma maneira comum de implementar ambas técnicas é a utilização de tabelas: dado o estado corrente do diálogo e o token de entrada corrente, um item da tabela determina o próximo estado do diálogo e quaisquer ações que devam ser executadas. Embora isto seja bastante rápido, tende a ser ineficiente em espaço porque as tabelas costumam ser bastante esparsas.

Os algoritmos de conversão essencialmente compilam a tabela dentro dos tratadores de eventos. Em alguns diálogos isto pode implicar em uma economia de espaço. Ao receber um token, o tratador de eventos ativo realiza

as operações que seriam indicadas na tabela. A única operação que poderia ser considerada ineficiente seria a criação de novos tratadores de eventos. A parte mais demorada dessa operação é a alocação dinâmica de memória para as variáveis locais do tratador de eventos, o que não chega a afetar a eficiência da implementação. Segundo [GREEN 86], na prática o usuário não consegue distinguir entre um diálogo implementado por tratadores de eventos e um implementado por software específico para redes de transição ou gramáticas. Embora a implementação por tratadores de eventos não possa ser tão eficiente quanto o uso de técnicas de implementação específicas, ela não é significativamente mais lenta.

3.3.4. Consequências práticas

Os resultados teóricos acima tem pelo menos duas implicações práticas.

Primeiramente, se for criado um UIDS com apenas uma notação de projeto, o modelo de eventos é a melhor escolha, supondo que um dos objetivos desse UIDS seja a possibilidade de descrever a maior quantidade possível de interfaces de usuário diferentes.

Em segundo lugar, um UIDS baseado no modelo de eventos é capaz de atender aos três modelos de diálogo, já que para cada interface de usuário descrita através de uma RTR ou uma GLC existe uma descrição equivalente usando a notação de eventos. Portanto, o modelo de eventos pode servir como base da representação interna do componente de controle de diálogo da interface de usuário. Caso os outros modelos sejam usados na descrição da interface, eles podem ser convertidos automaticamente em descrições usando o modelo de eventos. Desta maneira, o sistema de execução necessário para o UIDS precisa ser implementado para apenas um modelo, enquanto o projetista de interfaces pode utilizar o modelo que lhe for mais conveniente para a interface que estiver especificando.

CAPÍTULO 4

FERRAMENTAS PARA O DESENVOLVIMENTO DE INTERFACES DE USUÁRIO

... a tool should be as simple as possible, but no simpler. A tool is counterproductive when a large part of the entire project is taken up with mastering the tool.

Niklaus Wirth, [WIRTH 85]

Já foi desenvolvida uma quantidade significativa de ferramentas para tornar as interfaces de usuário mais baratas e fáceis de projetar e implementar. Algumas dessas ferramentas tiveram um sucesso muito grande. Por exemplo, no caso da ferramenta **MacApp** da Apple Computer, [SCHMUCK 86] relata uma diminuição do tempo de desenvolvimento de interfaces em um fator de quatro ou cinco vezes.

Neste Capítulo apresentamos uma resenha de algumas das ferramentas para o desenvolvimento de interfaces. Antes de descrever as ferramentas propriamente ditas, apresentamos alguns critérios de classificação dessas ferramentas. Algumas partes deste Capítulo são baseadas em [MYERS 89].

4.1. Tipos de ferramentas de interface

As ferramentas para o desenvolvimento de interfaces de usuário podem ser divididas em duas categorias gerais:

- Kits para o Desenvolvimento de Interfaces de Usuário (abreviado *UIDK*, do inglês, "User Interface Development Kits"), e
- Sistemas para o Desenvolvimento de Interfaces de Usuário (abreviado *UIDS*, do inglês, "User Interface Development Systems").

Um *UIDK* é uma biblioteca de técnicas de interação. Uma técnica de interação é uma maneira de usar um determinado dispositivo físico (como por exemplo um teclado, um mouse, um tablet ou um potenciômetro) para entrar com um valor (tal como um comando, número, percentual, posição ou nome), mais a realimentação correspondente a ser exibida na tela. Menus, barras de deslocamento e botões de comando operados com o mouse são exemplos de técnicas de interação.

Um *UIDK* é utilizado pelo programador escrevendo código que chama e organiza as técnicas de interação usadas na interface da aplicação. Em geral, os *UIDKs* não fornecem um auxílio elaborado ao projeto de interfaces ou à especificação do seqüenciamento das interações e do controle do diálogo.

Já um *UIDS* é um conjunto integrado de ferramentas que ajudam o programador a criar e gerenciar vários aspectos das interfaces de usuário, não apenas as técnicas de interação propriamente ditas. Assim um *UIDS* pode incluir, por exemplo, um componente de controle de diálogo ou ferramentas de auxílio ao projeto da interface. De forma geral, um *UIDS* ajuda no projeto

e na implementação da interface da aplicação, sendo portanto bem mais abrangente que um UIDK.

4.2. Comunicação entre ferramenta e aplicação

Uma outra classificação das ferramentas de interface de usuário pode ser obtida analisando a comunicação entre a ferramenta e o código da aplicação durante a execução do programa aplicativo já terminado:

- *Controle interno*, nome dado à situação em que o controle é da aplicação. Esta simplesmente chama os procedimentos da interface de usuário quando deseja utilizar alguma das técnicas de interação disponíveis. Este modelo é usado pelos UIDKs.
- *Controle externo*, denominação da situação em que o controle é da ferramenta. Neste caso são os procedimentos da interface de usuário que chamam a aplicação quando o usuário solicita a execução de um comando. Este modelo é usado pela maioria dos UIDSs, já que permite à ferramenta ter melhor controle sobre o escalonamento e o seqüenciamento da execução das tarefas que compõem a aplicação.

Analisando a comunicação do UIDS com a aplicação podemos ainda subclassificar o controle externo:

- *Procedimentos chamados** pelo UIDS, método bastante popular que consiste na chamada direta de procedimentos da aplicação pelo UIDS. É necessária a existência de um mecanismo que permita à aplicação informar o UIDS quais os nomes dos procedimentos a

---

* "Callback procedures", em inglês.

serem chamados. Desta maneira a aplicação acaba sendo um conjunto de procedimentos que são chamados pela interface. Esta é provavelmente a técnica mais direta e eficiente.

- *Memória compartilhada*, onde o UIDS e a aplicação podem ambos monitorar os dados compartilhados (para verificar eventuais mudanças de valores), ou o UIDS e a aplicação podem se notificar mutuamente através de um outro mecanismo de comunicação entre processos quando houver alterações nos dados compartilhados.
- *Mensagens ou eventos*, caso em que o UIDS comunica à aplicação da ocorrência de um evento, possivelmente acompanhado de argumentos (por exemplo, o evento clique de mouse, acompanhado das coordenadas da posição do mouse no momento do clique). Embora esta técnica talvez seja melhor estruturada, ela é menos eficiente que o uso de memória compartilhada ou a chamada direta dos procedimentos da aplicação.

Finalmente, pode existir o controle misto, onde (ao longo da execução da aplicação) tanto o UIDS como a aplicação podem deter o controle da execução.

Uma outra questão relativa à comunicação entre a ferramenta e a aplicação é a frequência e a quantidade de informação que precisa ser transmitida. De um ponto de vista ideal, essa comunicação deve ser pouco freqüente e simples, visando uma maior separação entre a interface de usuário e a lógica da aplicação.

No entanto, o uso da realimentação semântica exige uma comunicação freqüente entre a interface e a aplicação. Nas interfaces de manipulação direta, essa comunicação pode chegar a ser necessária até 60 vezes por segundo para, por exemplo, determinar se as posições pelas quais um objeto sendo arrastado pelo mouse são corretas ou não para posicionar o objeto. No entanto, a quantidade de informação que circula a cada comunicação é bastante limitada.

4.3. Tipos de usuários dos UIDSs

Existem basicamente quatro categorias de pessoas envolvidas na utilização de qualquer UIDS:

- O *criador* da ferramenta, ou seja o projetista e implementador do próprio UIDS.
- O *projetista* da interface que será criada com a ajuda do UIDS, que pode ser um analista, um programador ou um artista gráfico.
- O *programador* que cria a aplicação que utilizará a interface criada pelo projetista.
- O *usuário* da aplicação criada pelo programador.

Embora a classificação acima considere cada papel como sendo desempenhado por uma pessoa diferente, pode haver mais de uma pessoa desempenhando cada papel, e uma mesma pessoa pode desempenhar vários desses papéis.

4.4. User Interface Development Kits

A maioria dos sistemas de janelas e dos UIDSs são fornecidos junto com um UIDK contendo rotinas que podem ser usadas pelas aplicações. Tipicamente estão incluídos, entre outros, vários tipos de menus e barras de deslocamento.

A forma mais convencional de um UIDK é uma coleção de procedimentos a serem chamados pelas aplicações. Um exemplo é o **Macintosh Toolbox**, descrito por exemplo em [APPLE 85]. Outra forma de UIDK é baseada no uso de um estilo de programação orientado a objetos, incluindo herança. A herança facilita o trabalho de adaptação das técnicas de interação a cada caso ("personalização"). Um exemplo é o **Toolkit X.11** do X Window System, descrito em [CORMACK 88].

Existem dois UIDKs, chamados **Grow**, descrito em [BARTH 86], e **Coral**, descrito em [SZEKELY 88], que adicionam restrições* em relação aos toolkits orientados a objeto. Essas restrições permitem que o projetista especifique o relacionamentos entre os objetos, deixando por conta da ferramenta a manutenção desses relacionamentos. Por exemplo, se o projetista especificar que uma linha começa na metade de um lado de um retângulo e termina no meio de um lado de outro retângulo, a linha será movimentada automaticamente pela ferramenta se qualquer um dos retângulos mudar de lugar.

As desvantagens de usar UIDKs são que estes fornecem um conjunto limitado de estilos de interação e são freqüentemente caros de criar e difíceis de usar. Um UIDK tipicamente inclui centenas de procedimentos para implementar uma variedade de técnicas de interação. Muitas vezes não fica claro como usar os procedimentos para obter a interface de usuário desejada.

---

* "Constraints", em inglês.

4.5. User Interface Development Systems

Os problemas com os UIDKs levaram à criação dos UIDSs. De fato, o UIDS **MacApp** foi criado pela Apple depois de descobrir que as pessoas estavam tendo dificuldades no uso do **Macintosh Toolbox**. Os UIDSs ajudam o projetista da interface a combinar e seqüenciar as técnicas de interação. Alguns UIDSs ajudam o projetista a criar seus próprios UIDKs, enquanto outros ajudam a utilizar elementos de UIDKs predefinidos.

Os UIDSs podem ser classificados pela maneira com que permitem ao projetista especificar a interface. Alguns UIDSs usam linguagens próprias, alguns usam especificações gráficas e outros geram a interface automaticamente a partir da especificação funcional da aplicação. Ainda, alguns UIDSs usam técnicas diferentes para especificar partes diferentes da interface. Adotamos neste trabalho a classificação de [MYERS 89] (veja as seções 4.6 a 4.8).

4.5.1. Funções de um UIDS

Um UIDS abrangente deve gerenciar todos os aspectos da interface, incluindo todas as partes visíveis da tela e todos os aspectos do diálogo entre o usuário e a aplicação. O UIDS deve

- gerenciar o mouse e outros dispositivos de entrada;
- validar as entradas do usuário;
- tratar os erros do usuário;
- processar o cancelamento de comandos a pedido do usuário;

- fornecer realimentação apropriada para que o usuário perceba que sua solicitação foi recebida pela aplicação;
- fornecer um sistema de ajuda e de mensagens de prontidão;
- atualizar a tela quando os dados da aplicação mudam;
- notificar a aplicação quando o usuário atualiza dados da aplicação;
- gerenciar a edição de campos de entrada digitada;
- tornar o código da aplicação independente das características físicas da tela de vídeo usada como dispositivo de saída;
- avaliar a interface automaticamente e propor melhorias, ou pelo menos fornecer informação que permita ao projetista avaliar a interface; e
- permitir ao usuário personalizar a interface.

4.5.2. Componentes de um UIDS

Para fornecer as funções acima descritas, um UIDS pode conter

- um UIDK;
- um componente de controle do diálogo que gerencia a seqüência de eventos e as técnicas de interação;
- um modelo de desenvolvimento* da programação, que ajuda e orienta a estrutura da interface e a semântica da aplicação;

---

* "Application Framework", em inglês.

- um editor gráfico para especificar as características e a posição dos elementos da interface na tela; e
- um componente de análise que avalia a interface automaticamente (baseado em determinadas regras ou orientações previamente estabelecidas) ou armazena informação (por exemplo, os eventos) para avaliação posterior por parte do projetista.

4.6. UIDSs baseados em linguagens

Na grande maioria dos UIDSs, o projetista especifica a interface através de uma linguagem especialmente definida pelo criador do UIDS. Esta linguagem pode adotar muitas formas diferentes: desde simples hierarquias de menus, passando por redes de transição, gramáticas livres de contexto, linguagens de tratamento de eventos, linguagens declarativas, até linguagens orientadas a objetos. Na maioria dos sistemas o projetista da interface usa a linguagem para especificar a sintaxe da interface – a seqüência correta de ações de entrada e saída.

4.6.1. Hierarquias de menus

Uma das formas mais simples de UIDS é baseada em uma hierarquia ou rede de menus. Quando o usuário seleciona uma opção em um menu, outro menu é exibido.

O UIDS **Tiger** [KASIK 82] permite o uso de uma rede bastante complexa de menus que inclui a possibilidade de pular de nível e abortar. Muitos sistemas de hipertexto (veja por exemplo [CONKLIN 87]), como por exemplo o programa **HyperCard** da Apple, também podem ser considerados UIDS baseados em redes de menus.

4.6.2. Redes de transição

Os primeiros sistemas que podem ser chamados de UIDSs, descritos em [NEWMAN 68] e [PARNAS 69], eram baseados em redes de transição. Newman utilizou um autômato finito para implementar um UIDS que tratava apenas entrada textual. Muitas das técnicas utilizadas em sistemas mais modernos já eram usadas por Newman: linguagens diferentes para definir a interface e a semântica, análise sintática dirigida por tabelas e independência dos dispositivos físicos.

Um dos defensores das redes de transição é Robert Jacob [JACOB 86], que afirma que as redes de transição são melhores que as gramáticas livres de contexto porque exibem a seqüência temporal de eventos explicitamente. Jacob utiliza na realidade redes de transição recursivas.

Rapid/USE [WASSERMANN 82] é um outro sistema que usa redes de transição semelhantes às de Jacob, exceto que as primitivas para gerar saída são mais poderosas. **Rapid/USE** é na realidade parte de um sistema maior de engenharia de software.

Usando redes de transição, a conexão entre a interface e a aplicação é implementada através de variáveis globais. Todos os estados devem possuir arestas para explicitar todas as entradas erradas e para os comandos universais como cancelamento e ajuda.

UIDSs baseados em redes de transição são os mais úteis quando a interface precisa fazer muita análise sintática ou quando possui muitos modos (cada estado é na realidade um modo — veja a nota de rodapé da página 3). No entanto, a maioria dos sistemas interativos são praticamente amodais, dando ao usuário muitas opções a cada momento. Como isto implica em um número grande de arestas saindo de cada estado, este tipo de UIDSs não tem tido sucesso com este tipo de interfaces.

Outro problema com as redes de transição é que não permitem lidar com interfaces que permitam ao usuário operar concorrentemente sobre vários objetos (possivelmente usando vários dispositivos de entrada diferentes). Ainda, os diagramas tornam-se muito confusos quando usados com interfaces grandes.

Jacob inventou um novo formalismo, apresentado em [JACOB 86], que reconhece estes problemas mas tenta manter a clareza das redes de transição. O novo formalismo combina as redes de transição com uma forma de linguagem de eventos. Vários diagramas podem estar ativos ao mesmo tempo e o controle pode ser transferido de um para outro como se fossem corotinas. O UIDS resultante, denominado **Interaction Objects**, permite criar interfaces de manipulação direta.

4.6.3. Gramáticas livres de contexto

A maioria dos UIDS baseados em gramáticas são baseados em geradores de analisadores sintáticos. Nesses casos, o projetista pode especificar a sintaxe da interface em alguma variante da notação BNF.

O primeiro uso de gramáticas livres de contexto no projeto e implementação de interfaces de usuário é difícil de determinar. Em meados da década de 70 um número razoável de grupos de pesquisa tentou usar geradores de analisadores sintáticos disponíveis, tal como YACC [JOHNSON 77], para produzir interfaces de usuário para programas gráficos.

Na maioria dos casos esses experimentos fracassaram, devido aos problemas encontrados no tratamento de erros. As rotinas de tratamento de erros fornecidas com os geradores de analisadores sintáticos foram projetadas para uso com linguagens de programação. A dificuldade básica reside em continuar a fornecer realimentação sintática e semântica ao usuário enquanto a reversão do erro estiver em andamento, obtendo ao mesmo tempo a informação necessária para retornar o analisador sintático a um estado "normal".

Entre os primeiros UIDSs baseados em gramáticas podemos citar as duas versões do sistema **SYNICS**, respectivamente descritas em [EDMONDS 77] e [GUEST 82], e o trabalho de Hanau e Lenorovitz [HANAU 80].

Syngraph [OLSEN 83] gera programas de interface em Pascal contendo um analisador sintático para o diálogo a partir de uma descrição escrita como uma gramática formal em notação BNF estendida. **Syngraph** gerencia menus, entrada textual e alguns outros tipos predefinidos de técnicas de interação.

Syngraph é capaz de tratar a reversão de erros semânticos e o comando de cancelamento no nível semântico. Ainda, é capaz de decidir qual objeto selecionar se houver vários objetos na posição indicada na tela.

No entanto, **Syngraph** não é capaz de produzir realimentação semântica ou valores padrões*, já que as rotinas da aplicação não tem como interferir na análise sintática.

4.6.4. Linguagens de tratamento de eventos

No caso das linguagens de eventos, os tokens de entrada são considerados como eventos que são imediatamente enviados a tratadores de eventos. Estes tratadores podem gerar eventos de saída, mudar o estado interno do sistema e chamar rotinas da aplicação.

O modelo de eventos foi motivado pela pesquisa efetuada com Smalltalk [GOLDBERG 83]. Um dos primeiros usos de eventos e tratadores de eventos é descrito em [ELSHOFF 74]. Outra implementação é apresentada em [GREEN 79], a qual evoluiu para ser usada no UIDS **University of Alberta** [GREEN 85].

---

* "Default", em inglês.

Outros dois UIDSs baseados no modelo de eventos são **ALGEA** e **Sassafras**. **ALGEA** [FLECHIA 87] usa uma linguagem de eventos que é uma extensão de Pascal. O projetista programa a interface como um conjunto de pequenos tratadores de eventos, que **ALGEA** transforma em código convencional.

Sassafras [HILL 86] usa uma idéia similar, mas com uma sintaxe completamente diferente. Também adiciona variáveis locais chamadas "flags" para ajudar a especificar o controle de fluxo. **Sassafras** é especialmente adequado para interfaces que utilizam diversos dispositivos de entrada concorrentemente. Também é capaz de atender às interfaces de manipulação direta porque promove uma comunicação eficiente e freqüente entre as técnicas de interação e o programa aplicativo.

Squeak [CARDELLI 85] é uma linguagem textual para a programação de interfaces baseadas em mouse que explora a concorrência. Os processos de **Squeak** são similares a tratadores de eventos e as mensagens enviadas a seus processos são semelhantes a eventos (a diferença é que as mensagens não são enviadas enquanto não houver um processo pronto para recebê-las).

Squeak permite o uso de vários dispositivos de entrada ativos de forma concorrente. Os eventos de entrada primitivos são transições de estado dos botões do mouse, teclas apertadas no teclado, movimentos do mouse ou de um outro dispositivo apontador, e eventos gerados a partir de um temporizador*. **Squeak** transforma o programa em um autômato de estados. Apesar de fornecer uma notação compacta para especificar interfaces complexas com dependências temporais, **Squeak** é uma linguagem bastante difícil de se programar.

Quase todos os UIDSs nesta categoria são explicitamente projetados para lidar com múltiplos processos. Pesquisas demonstraram que os usuários

---

* "Timer", em inglês.

podem ser mais produtivos quando operam vários dispositivos de entrada simultaneamente [BUXTON 86]. Frequentemente também é mais fácil utilizar múltiplos processos para programar múltiplas interações das quais o usuário pode escolher qual utilizar.

A principal desvantagem das linguagens de eventos é o fato de que é bastante difícil criar código correto porque o controle de fluxo não é localizado. Em alguns casos, pequenas alterações em uma parte do programa podem afetar muitas outras partes. Muitas vezes também é difícil para o projetista entender o código quando este adquire um tamanho significativo.

4.6.5. Linguagens declarativas

As linguagens declarativas definem *o que* deve ocorrer ao invés de *como* deve ocorrer. As interfaces descritas por linguagens declarativas são geralmente baseadas em formulários: o usuário digita texto em campos ou seleciona opções através de menus ou botões. Às vezes também existem áreas de saída gráficas reservadas para a aplicação. A aplicação é conectada à interface através de variáveis globais, modificadas e consultadas tanto pela aplicação como pela interface.

Cousin [HAYES 85] e **Domain/Dialogue** [SCHULER 85] são UIDS baseados em linguagens declarativas. A Apollo, fabricante de estações de trabalho posteriormente incorporada pela Hewlett-Packard, criou um novo UIDS baseado em **Domain/Dialogue**, chamado **Open Dialogue** que executa sob o X Window System.

A vantagem dos UIDSs baseados em linguagens declarativas é que eles libertam o projetista da preocupação com a sequência de eventos, passando a concentrar-se sobre a informação que entra e sai da aplicação.

A desvantagem é que servem apenas para interfaces baseadas em formulários com um conjunto fixo e preprogramado de técnicas de interação; outros tipos de interação tem que ser codificados manualmente usando as áreas gráficas reservadas para a aplicação.

4.6.6. Linguagens orientadas a objetos

Os sistemas desta categoria formam uma categoria nova de UIDS. Eles fornecem um modelo de desenvolvimento para a aplicação, dentro do qual o projetista programa a interface. Tipicamente, existem classes de alto nível que descrevem o comportamento normal da interface. O projetista especializa essas classes para lidar com o comportamento específico da interface sendo criada, valendo-se do mecanismo de herança embutido nas linguagens orientadas a objetos.

Entre os UIDS baseados na orientação a objetos podemos citar **MacApp** da Apple [SCHMUCK 86], que é programado usando a linguagem Object Pascal, e **GWUIMS** [SIBERT 86] da George Washington University, que utiliza Flavors, uma versão orientada a objetos de Lisp.

Estes sistemas são capazes de lidar com interfaces interativas de manipulação direta, já que a ligação entre a entrada e a saída pode ser modificada pela aplicação para definir a semântica. Embora estes UIDSs simplifiquem a criação de interfaces, deve-se lembrar que são ambientes de programação, e como tais são inacessíveis a projetistas que não sejam programadores.

4.7. UIDSs baseados em especificações gráficas

UIDSs gráficos permitem que a interface seja definida, pelo menos parcialmente, posicionando objetos na tela com um mouse. Os proponentes destes UIDSs sustentam que já que a apresentação visual da interface é um dos seus aspectos mais importantes, uma ferramenta gráfica é a maneira mais apropriada de especificar essa apresentação.

Esta técnica é naturalmente muito mais fácil de ser usada pelo projetista. Alguns destes sistemas, como por exemplo **Menulay**, **Trillium** e **Peridot**, descritos abaixo, podem ser usados também por projetistas que não sejam programadores.

Os UIDSs desta categoria têm três desvantagens em função do uso de gráficos: a construção do próprio UIDS é mais complicada, o UIDS resultante permite o uso de uma variedade limitada de interfaces e o tratamento da ajuda, das mensagens de prontidão e do cancelamento de comandos fica por conta da aplicação.

Alguns UIDSs desta categoria permitem que o projetista posicione na tela as técnicas de interação, tais como menus, botões e barras de deslocamento. Estes sistemas geralmente permitem que certas áreas da tela sejam reservadas para a entrada e/ou saída de gráficos específicos da aplicação. O projetista deve definir os procedimentos a serem executados quando o usuário usa cada técnica de interação. Nesta categoria podemos citar o **DialogEditor** [CARDELLI 87] e o **Interface Builder**, da empresa Expertelligence para Macintosh.

Outros sistemas gráficos, tais como **Menulay** [BUXTON 83], **Trillium** [HENDERS 86] e o **HyperCard** da Apple, organizam a interface como uma rede de páginas ou frames praticamente estáticas. Cada página contém texto, gráficos e técnicas de interação, além de comandos que permitem mudar de página corrente. Normalmente as técnicas de interação são codificadas utilizando-se uma linguagem de programação convencional.

Menulay [BUXTON 83] permite ao projetista posicionar texto, ícones, botões e potenciômetros na tela exatamente da maneira que serão vistos pelo usuário quando a aplicação estiver sendo executada. Cada item ativo é associado a uma rotina semântica da aplicação, chamada quando o usuário selecionar o item com um dispositivo apontador. As rotinas semânticas são escritas usando uma linguagem de programação convencional. **Menulay** gera tabelas e código que, ligados ao seu sistema de execução, executam a interface.

Menulay é suficientemente geral para ser capaz de gerar a sua própria interface. No entanto, a sua utilização de uma estrutura orientada por tabelas limita as possibilidades de comunicação entre a interface e o nível semântico da aplicação, prejudicando a qualidade da realimentação semântica.

Trillium [HENDERS 86] é bastante parecido com **Menulay**. Este UIDS foi criado para auxiliar o projeto de painéis de controle de máquinas fotocopadoras. Uma vantagem de **Trillium** sobre **Menulay** é o fato de interpretar a especificação, o que permite executar a interface enquanto está sendo projetada. **Trillium** também separa o comportamento da interação da apresentação gráfica, de forma que o projetista pode mudar a apresentação sem modificar o comportamento. No entanto, **Trillium** praticamente não permite mudanças de página (ou frame) corrente, já que esse recurso não é necessário para fotocopadoras.

O sistema de hipertexto **HyperCard** da Apple para computadores Macintosh também pode ser enquadrado nesta categoria. Ele permite a especificação gráfica (e programação, numa linguagem chamada HyperTalk) de páginas estáticas. Usando o editor, o projetista pode definir texto e gráficos na página corrente, junto com botões que geram transições para outras páginas quando acionados pelo usuário.

Grins [OLSEN 85] combina um analisador sintático com um sistema de restrições sobre a entrada e saída para tratar a realimentação semântica. Um editor gráfico permite que o projetista posicione as técnicas de interação (menus, ícones e áreas de texto) com um mouse.

Peridot [MYERS 86] difere de todos os UIDS anteriores pelo fato de permitir ao projetista criar as próprias técnicas de interação graficamente. O projetista pode usar as primitivas gráficas (como retângulos, círculos, linhas e textos) para construir menus, barras de deslocamento e botões. **Peridot** generaliza as ações do projetista para criar procedimentos orientados a objeto semelhantes àqueles encontrados em UIDKs. **Peridot** é suficientemente geral para criar a sua própria interface e a maior parte das técnicas de interação definidas no **Macintosh Toolbox**.

4.8. UIDSs com geração automática

Uma nova categoria de UIDSs tenta gerar a interface automaticamente a partir da especificação dos procedimentos semânticos da aplicação, permitindo eventualmente que o projetista modifique a interface depois para melhorá-la. Estes sistemas tentam resolver as dificuldades que são encontradas na utilização de outros tipos de UIDSs e a relutância das pessoas em utilizá-los [OLSEN 87].

Control-Panel Interface [FISHER 87] utiliza os tipos dos parâmetros dos procedimentos para criar uma interface gráfica (com botões para expressões booleanas, potenciômetros para valores inteiros, etc). O projetista pode usar os controles para modificar os valores e executar os procedimentos.

MIKE [OLSEN 86] é capaz de criar interfaces a partir de uma lista dos procedimentos da aplicação. A interface padrão criada inicialmente é baseada em menus. O projetista pode utilizar um editor gráfico para modificar a interface criada, modificando a estrutura do menu e substituindo nomes de comandos por ícones, entre outros. **MIKE** não permite a utilização de interações contínuas, como são usadas, por exemplo, no Macintosh, no Windows ou no Open Windows para modificar o tamanho de uma janela.

IDL [FOLEY 87] exige que a semântica da aplicação seja definida em uma linguagem especial, de sintaxe semelhante à Pascal. A partir dessa descrição o sistema gera automaticamente uma interface baseada em menus. Um aspecto interessante de **IDL** é que o projetista pode modificar a interface aplicando transformações. Por exemplo, uma transformação modifica a interface de maneira que o objeto seja selecionado antes da operação que será aplicada (e não depois). Outra transformação permite especializar

funções para tipos específicos (por exemplo, definindo apagar-triângulo a partir de apagar). IDL aplica as transformações garantindo que a interface resultante seja consistente.

4.9. Problemas das ferramentas de interface

Embora o uso de UIDKs esteja crescendo em função da popularização das interfaces gráficas em microcomputadores (Macintosh, Windows, Presentation Manager) e nas estações de trabalho (X System e derivados), a sua utilização e a dos UIDSs ainda é limitada em função de uma série de problemas:

- As ferramentas são difíceis de usar. Os UIDKs tipicamente contém centenas (quando não milhares) de procedimentos que interagem de maneira sutil. A maioria dos UIDSs exige que o projetista aprenda uma nova linguagem. Na prática, os projetistas programadores tem-se mostrado relutantes a aprender uma nova linguagem para especificar a interface [OLSEN 87]. Como estas linguagens são geralmente semelhantes a linguagens de programação, elas também não podem ser usadas por projetistas que não sejam programadores. UIDSs baseados em gráficos e de geração automática tentam resolver este problema, mas quase todos esses sistemas ainda são de caráter experimental.
- As especificações da interface de usuário são difíceis de entender e de editar. As linguagens usadas por muitos UIDSs para especificar a interface possuem uma estruturação pobre (do ponto de vista da engenharia de software). Elas usam variáveis globais, controle de fluxo não localizado e desvios incondicionais explícitos. Algumas linguagens gráficas, como as redes de transição, podem transformar-se em um emaranhado incompreensível de setas quando

a especificação da interface é grande. Conseqüentemente, pode ser muito difícil de entender ou modificar uma interface especificada através de um UIDSs. Este é um problema muito sério para uma tecnologia que pretende viabilizar o uso do projeto iterativo.

- A funcionalidade das ferramentas é limitada. A maioria dos UIDKs e UIDSs auxiliam apenas em uma parte do trabalho de projeto da interface. Embora em geral o tratamento de menus e barras de deslocamento seja excelente, o tratamento da exibição e manipulação dos dados da aplicação é bastante limitado. Poucos UIDSs permitem o uso de interfaces de manipulação direta porque estas exigem realimentação semântica, que nem sempre está disponível. UIDSs gráficos e de geração automática são mais fáceis de usar, mas têm funcionalidade mais restrita. UIDSs baseados em linguagens são mais gerais, mas mesmo assim apenas uma minoria permite o uso das interfaces de manipulação direta.
- As ferramentas não estão disponíveis e não são portáteis. As ferramentas de interface disponíveis comercialmente ainda são muito poucas, e a maioria destas tipicamente funciona apenas em um tipo específico de equipamento. Por exemplo, o **Macintosh Toolbox** e **MacApp** não podem ser usados a não ser que esteja sendo desenvolvido software para ser executado em equipamentos da família Apple Macintosh. O X Window System, seu Toolkit e UIDSs derivados oferecem a possibilidade de ter software de interface mais portátil. No entanto, os UIDS derivados tendem a usar recursos específicos dos equipamentos para os quais foram criados.
- As ferramentas não permitem a avaliação das interfaces. Pouquíssimas ferramentas fornecem algum tipo de auxílio para avaliar a interface. É necessária mais pesquisa a respeito de como o computador pode ser usado para essa avaliação antes que esse tipo de recurso seja prático. É necessário definir como o computador

pode avaliar a interface (a maioria das regras são informais e gerais demais para serem usadas por um computador) e como armazenar a informação apropriada eficientemente (guardar todas as teclas apertadas gera dados demais; ainda, este método não funciona bem com interfaces baseadas em mouse).

- As ferramentas são difíceis de serem construídas. Apesar da intensa pesquisa na construção de UIDKs e UIDSs, restam muitas dificuldades. As ferramentas realmente úteis na prática demandaram um esforço muito grande para serem produzidas.
- Os projetistas não desejam perder o controle. A maioria das ferramentas de interface forçam um estilo próprio de interface. A vantagem é que as aplicações terão interfaces similares e o projetista não precisa criar um estilo de interface para cada aplicação desenvolvida. A desvantagem é que alguns projetistas não querem perder a flexibilidade de que dispõem atualmente porque desejam um estilo de interface único e diferente para seus aplicativos. Este aspecto pode ser importante para diferenciar seus produtos dos oferecidos pela concorrência. Alguns UIDSs de pesquisa, como por exemplo **Peridot**, oferecem uma certa independência de estilo, mas isto complica a tarefa de projetar a interface. Provavelmente é impossível que uma ferramenta de interface de usuário seja totalmente independente de estilos se fornecer mais auxílio que uma linguagem de programação convencional.
- As ferramentas podem prejudicar a qualidade da interface. As ferramentas freqüentemente adicionam um nível de interpretação à execução da aplicação, o que pode diminuir o desempenho de maneira sensível. As ferramentas geralmente impedem aos projetistas de otimizar qualquer aspecto da interface manualmente, o que resulta em interfaces com desempenho pior do que se tivessem sido criadas de forma totalmente manual.

- A separação entre aplicação e interface é difícil. Uma pesquisa efetuada entre projetistas de interface [ROSSON 87] concluiu que 50% dos projetistas afirmaram que "a interface de usuário não tinha sido considerada distinta do restante do sistema durante o projeto; muitos projetistas pareciam ter séria dificuldade em sequer imaginar como a separação poderia ser aplicada ao sistema que eles tinham projetado, e alguns fizeram afirmações categóricas sobre a inconveniência ou impossibilidade de se efetuar uma tal distinção".

Apesar do sucesso limitado das ferramentas descritas e dos muitos aspectos a serem pesquisados ainda, o futuro das ferramentas de interface de usuário é brilhante. Não é óbvio, até o momento, qual enfoque será o mais frutífero nas pesquisas e no desenvolvimento comercial, mas é claro que existem uma necessidade crescente e demanda por esse tipo de ferramentas, de maneira que o ritmo de pesquisa e desenvolvimento dessas ferramentas provavelmente será acelerado [MYERS 89].

CAPÍTULO 5

MITUI, UM UIIDS BRASILEIRO

Neste Capítulo descreveremos uma ferramenta de interface de usuário projetada e implementada pelo autor na qualidade de sócio-diretor da empresa MBI - Mayer & Bunge Informática S/C Ltda. A marca do software é "MITUI - The first high-level CUA API®", que daqui em diante será denotada simplesmente como MITUI, omitindo inclusive o símbolo ® (que indica tratar-se de marca registrada), para tornar a leitura mais agradável.

5.1. O padrão CUA e o Windows

A IBM definiu em 1987 uma estratégia para o desenvolvimento de aplicações denominada SAA ("Systems Application Architecture"), com o objetivo de padronizar os componentes de software relativos a gerenciamento de banco de dados, teleprocessamento e interface de usuário, sob a justificativa de tornar tanto os produtores de aplicações como os usuários mais produtivos.

O componente da SAA relativo à interface de usuário é denominado CUA ("Common User Access"). Sua descrição definitiva foi publicada pela IBM somente em 1989 [IBM 89]. Este documento descreve os princípios

conceituais, o modelo de aplicação e as técnicas de interação que compõem o padrão CUA, usando como exemplo o ambiente gráfico Presentation Manager do sistema operacional OS/2, desenvolvido pela IBM em colaboração com a Microsoft. No entanto, não é feita nenhuma referência a qualquer técnica específica de implementação.

O padrão CUA essencialmente define um menu (de dois níveis) para a aplicação, denominado pela IBM de *barra de ações* ("action bar"), um conjunto de *janelas de documento* ("document windows") para que a aplicação exiba os seus dados, e *caixas de diálogo* ("dialog boxes") modais para a obtenção de valores do usuário. Todos esses elementos juntos compõem a *janela da aplicação* ("application window"). Pode haver várias destas últimas se o ambiente permitir a execução concorrente de aplicações.

A definição do padrão CUA produziu um impacto em todo o mercado de software, à semelhança do ocorrido com muitos outros padrões adotados pela IBM. Os grandes produtores (por exemplo, Lotus, Microsoft, Borland, Symantec, além da própria IBM) adequaram suas interfaces ao padrão. Ainda, o ambiente operacional Windows 3.0, lançado pela Microsoft em maio de 1990, herdou todas as técnicas de interação definidas pelo padrão CUA. O UIDK fornecido pela Microsoft aos produtores de aplicações para o ambiente Windows 3.0, denominado Windows Software Development Kit (abreviado SDK), inclui um exemplar de [IBM 89] como parte da sua documentação.

O SDK utiliza um modelo de mensagens (existem mais de 400 mensagens diferentes) para a comunicação entre o ambiente e as aplicações. Ainda, existem mais de 700 procedimentos à disposição das aplicações. Para uma descrição detalhada do SDK veja por exemplo [MICROSOFT 90].

O sucesso comercial do Windows (em número total de cópias vendidas já supera largamente a quantidade de equipamentos Macintosh comercializados pela Apple) popularizou tremendamente o padrão CUA. Junto com o lançamento da versão 3.1 do Windows, em março de 1992, a Microsoft anunciou a divulgação, ainda no decorrer de 1992, de um documento com a definição das extensões ao padrão CUA por ela utilizada

no Windows 3.1. A Mayer & Bunge Informática foi uma das poucas empresas no Brasil a ter acesso à versão preliminar desse documento, lançado em definitivo como livro pela Microsoft Press [MICROSOFT 92] no mês de junho.

A partir desta situação de mercado tornou-se clara a necessidade de um UIDS específico para o padrão CUA de interface que isolasse a aplicação do tratamento das técnicas de interação. A criação de MITUI é consequência desta constatação.

5.2. MITUI como UIDS

MITUI foi projetado entre abril e setembro de 1990. A construção da primeira versão, em inglês, terminou somente em outubro de 1991. Já a versão em português foi liberada apenas em março de 1992.

MITUI foi construído na forma de uma biblioteca de funções, em linguagem C e para o sistema operacional MS-DOS, que encapsula, entre outros aspectos, todas as técnicas de interação. O programador da aplicação não precisa construir as técnicas de interação nem precisa preocupar-se com as sequências de eventos. Ele precisa escrever apenas a semântica da aplicação. Neste sentido MITUI é um UIDS de geração automática (veja a Seção 4.8.).

5.3. Características da Interface

MITUI utiliza como dispositivos de entrada um teclado, incluindo teclas de função, e um mouse (embora não seja indispensável, a sua utilização é recomendada). Como dispositivo de saída MITUI utiliza um monitor de vídeo, preferencialmente colorido.

Figura 5.1 - Realimentação semântica de um comando demorado

5.3.1. Realimentação

MITUI gerencia automaticamente a realimentação, não apenas no nível léxico, mas também no sintático. Por exemplo, de acordo com o padrão CUA, todo nome de comando do menu deve ter uma letra destacada (para permitir ao usuário ter acesso à opção diretamente, combinando a tecla especial Alt com a da letra em destaque). Durante a execução do código correspondente a um comando de menu selecionado pelo usuário, MITUI indica a não disponibilidade temporária do menu tirando o destaque das letras dos nomes dos comandos. Caso a execução do comando seja demorada, a aplicação pode facilmente exibir ao usuário a proporção do trabalho já terminado, em paralelo à execução do próprio comando, valendo-se para isso de caixas de diálogo contendo áreas dinâmicas (veja a Figura 5.1).

Quanto à realimentação semântica, MITUI fornece uma variedade de recursos para a aplicação. Por exemplo, dependendo do contexto atual, a aplicação pode indicar a não disponibilidade temporária de certas opções de seu menu. As janelas são outro exemplo do tratamento que MITUI dá à realimentação semântica. MITUI gerencia automaticamente a exibição dos dados da aplicação (mesmo quando estejam sendo exibidas múltiplas janelas, eventualmente sobrepostas e/ou subdivididas), exigindo da aplicação apenas que gerencie esses dados. Quando um comando da aplicação modificar os dados, basta que a aplicação notifique esse fato ao MITUI.

5.3.2. Sistema de ajuda e mensagens de prontidão

MITUI fornece um sistema de ajuda para as aplicações (veja a Figura 5.2). Esse sistema apresenta características de hipertexto: cada tópico (unidade de texto) de ajuda possui ligações com outros tópicos. O usuário pode facilmente navegar de um tópico para outro relacionado, e voltar ao tópico anterior.

Figura 5.2 - Um exemplo do sistema de ajuda de MITUI

Uma característica original de MITUI é a disponibilidade automática, para qualquer aplicação, de um conjunto de mais de 125 tópicos de ajuda. Estes tópicos descrevem as características das aplicações definidas pelo padrão CUA e todas as técnicas de interação.

A aplicação pode fornecer tópicos adicionais que descrevem, por exemplo, a semântica da aplicação. Para tanto, MITUI fornece um compilador de textos de ajuda, denominado **MITUI Help Maker**. A partir de um arquivo de texto normal, contendo os nomes e o texto dos tópicos, a compilação gera um arquivo no formato binário usado durante a execução. O relacionamento entre os tópicos é gerado automaticamente durante a compilação.

Durante a execução da aplicação, o usuário não tem como diferenciar os tópicos fornecidos automaticamente por MITUI daqueles que foram fornecidos pela aplicação. Embora permaneçam fisicamente em arquivos separados, os tópicos são armazenados em uma única estrutura de dados durante a execução da aplicação.

O sistema de ajuda pode ser acionado pelo usuário através do comando **Ajuda** do menu (qualquer aplicação desenvolvida com MITUI dispõe desse comando no menu automaticamente). Existem nesse menu várias opções, como escolher um tópico de ajuda do índice de tópicos disponíveis, achar os nomes dos tópicos contendo uma certa sequência de caracteres, ter acesso direto ao tópico de ajuda que detalha o funcionamento do próprio sistema de ajuda, obter a identificação da aplicação (acionada automaticamente quando a execução da aplicação é iniciada), e obter ajuda contextual.

A ajuda contextual também pode ser obtida, a qualquer momento, através da tecla de função *F1*. O contexto corrente é definido por MITUI de acordo com o pseudocódigo apresentado na Figura 5.3. Quando o sistema de ajuda é acionado, ele exibe o texto do tópico correspondente ao contexto corrente, ou, se esse tópico não existir, informa o fato ao usuário e lhe apresenta o índice dos tópicos disponíveis.

```
se há um diálogo em andamento
    contexto := título do diálogo corrente
senão se há uma opção de menu selecionada
    contexto := nome do comando selecionado
senão se há uma janela ativa
    contexto := palavra indicada pelo cursor
senão
    contexto atual não está definido
```

Figura 5.3 - Determinação do contexto de ajuda

Os menus de MITUI são projetados para usuários de todos os níveis de experiência. Usuários novatos podem navegar através das opções definidas pela aplicação, recebendo ainda uma curta mensagem de ajuda na área de status da aplicação. Usuários intermediários podem usar as letras em destaque para efetuar o acesso às opções sem navegar até elas, enquanto usuários experientes podem usar as teclas aceleradoras para acionar comandos sem precisar sequer acionar o menu. Como todos os métodos estão disponíveis permanentemente (o usuário não precisa indicar o método que deseja usar), métodos diferentes podem ser usados para partes diferentes da aplicação, conforme o nível de experiência do usuário com cada uma delas.

5.3.3. Reversão de Erros

MITUI fornece uma variedade de estratégias para a reversão de erros. MITUI trata automaticamente os erros léxicos. As entradas textuais possuem um complexo sistema de edição, que permite apagar qualquer parte da entrada (inclusive vários caracteres ao mesmo tempo) ou restaurar o valor inicial.

Os erros sintáticos também são tratados automaticamente. Por exemplo, MITUI determina automaticamente quais eventos de entrada são legais a cada instante.

Já o tratamento de erros semânticos precisa da cooperação da aplicação ou é de responsabilidade da mesma. Um exemplo de cooperação é a validação de entradas textuais nas caixas de diálogo (a aplicação deve fornecer um predicado a respeito da entrada). Já a reversão da ação de um comando é de responsabilidade da aplicação, que pode fornecê-la através de um outro comando no menu (geralmente chamado "Undo", 'Desfazer' ou 'Reverter' em inglês).

Ainda, MITUI trata erros cometidos pelo programador. Todos os argumentos que a aplicação fornece a MITUI são validados na medida do possível. Caso seja detectado um erro, MITUI gera um erro de execução, informando ao programador o tipo e o código do erro detectado. O código do erro remete à documentação impressa, que descreve as possíveis causas do erro e correspondentes soluções mais detalhadamente.

5.3.4. Tempo de resposta

MITUI foi projetado para ser usado inclusive em equipamentos com relativamente pouca capacidade de processamento, como por exemplo PCs do tipo XT. O tempo de resposta nesse tipo de equipamento é inferior a meio segundo para a absoluta maioria das operações. Em equipamentos mais potentes, o tempo de resposta não é perceptível.

Para atingir esses tempos de resposta, MITUI utiliza acessos de baixo nível aos dispositivos de entrada e saída. A tela é operada no modo texto (o que viabiliza o uso até mesmo com placas de vídeo monocromáticas sem nenhum recurso gráfico).

5.3.5. Consistência

MITUI garante automaticamente uma quantidade grande de aspectos de consistência da interface, como por exemplo a distribuição das partes da

tela, a apresentação de menus, diálogos e janelas, as técnicas de interação e o significado dos eventos de entrada.

No entanto, MITUI não pode garantir, por exemplo, que o posicionamento dos comandos no menu ou dos controles de interação de um diálogo apareçam sempre na mesma posição, porque estes são construídos dinamicamente durante a execução da aplicação (caso contrário o menu não poderia ser adaptado ao estado atual da aplicação: por exemplo, o menu de um depurador contendo os nomes das rotinas que foram ativadas até um certo ponto da execução do programa precisa ser determinado dinamicamente).

5.3.6. Visual estruturado

MITUI segue fielmente o padrão CUA na estruturação do visual da aplicação. O topo da área de aplicação apresenta o menu, ao pé da mesma área é apresentado o status da aplicação, e entre essas duas áreas são apresentadas as janelas contendo os dados da aplicação. MITUI utiliza um conjunto de cores diferenciado para cada parte do visual da aplicação.

O modelo de aplicação com múltiplas janelas é denominado *Multiple Document Interface* (MDI, ou Interface de Múltiplos Documentos) no padrão CUA (veja a Figura 5.4). A posição e o tamanho das janelas pode ser especificado pela aplicação, mas o usuário pode modificá-los a qualquer momento. Embora o código da aplicação seja totalmente independente desses valores, MITUI fornece esses valores à aplicação se esta os solicitar. Assim, a aplicação pode, por exemplo, maximizar o tamanho de uma janela durante a execução de um comando de substituição, restaurando-a ao tamanho e posição originais logo a seguir.

Figura 5.4 - O visual de MITUI com várias janelas e um diálogo

MITUI utiliza nas janelas um modelo de exibição seletiva baseado no conceito de *dados virtuais*. Cada janela apresenta, a cada momento, uma parte (ou várias partes, se a janela tiver sido subdividida em painéis) do total de dados a ela destinados. Cabe à aplicação decidir qual parte do total dos dados deve residir em memória (daí o nome de dados **virtuais**). MITUI precisa saber apenas o tamanho total dos dados virtuais para poder gerenciar automaticamente as barras de deslocamento da janela, que indicam ao usuário a posição e o tamanho relativos dos dados exibidos.

5.3.7. Comandos fornecidos por MITUI para a aplicação

Além do sistema de ajuda, já comentado, MITUI fornece automaticamente um menu de controle para a aplicação. Este menu de controle fornece comandos para encerrar a aplicação e para o usuário personalizar alguns aspectos da interface.

A personalização disponível depende da configuração exata dos dispositivos de entrada e saída disponíveis (monitores monocromáticos ou coloridos, resolução da placa de vídeo e da presença ou não de mouse), mas basicamente o usuário pode personalizar as cores utilizadas para todas as partes do visual da aplicação (a Figura 5.5 exibe um exemplo da personalização de cores), a sensibilidade do mouse ao movimento físico e o uso de realimentação sonora no caso de erros.

5.4. Técnicas de interação implementadas

Todas as técnicas de interação implementadas pelo UIDS MITUI são baseadas em [IBM 89] (definição original do padrão CUA). Todos os recursos podem ser utilizados através do teclado e do mouse. MITUI gera os tokens de entrada da aplicação em função da atividade do usuário.

No caso do teclado, MITUI reconhece não apenas os caracteres ASCII e a extensão desse sistema de codificação definida pela IBM para o PC (códigos de 128 a 255), mas também todas as combinações com as teclas de status (*Shift*, *Ctrl*, *Alt*, *Num Lock*, *Caps Lock* e *Scroll Lock*) reconhecidas pelo próprio firmware do teclado (e a BIOS do PC), mais as teclas especiais dos teclados IBM, como por exemplo as teclas de função e de movimentação do cursor. Assim o usuário pode gerar tokens como *K*, *Alt+K*, *Shift+Ctrl+K*, *Alt+F4* ou *Ctrl+End*, onde o sinal + indica que as teclas relacionadas devem ser pressionadas simultaneamente.

Com o mouse, o usuário pode utilizar os botões esquerdo e direito, separadamente ou ao mesmo tempo. Os tokens correspondentes a "clique" (pressionar e soltar) os botões do mouse podem ser modificados pelas teclas de estado *Shift*, *Ctrl* e *Alt*. Existem duas operações especiais com o mouse, tratadas como tokens diferentes: a primeira, denominada *clique duplo*^{*}, é gerada apertando o botão esquerdo do mouse duas vezes seguidas em um curto espaço de tempo; a segunda operação especial, denominada *arrasto*^{**}, é gerada pressionando o botão esquerdo do mouse e mantendo-o pressionado

---

* "Double click", em inglês.

** "Drag", em inglês.

enquanto o mouse é movimentado até a posição final da operação, instante em que o botão esquerdo é solto.

5.4.1. Menus

MITUI fornece um menu de dois níveis, isto é, o usuário efetua no máximo duas escolhas para acionar um comando da aplicação.

O primeiro conjunto de opções, chamado de barra de ações, é exibido de forma permanente na parte superior da tela (veja a Figura 5.6). A primeira opção (que dá acesso ao conjunto de opções de personalização da aplicação) e a última (que permite ao usuário operar o sistema de ajuda da aplicação) são fornecidas de forma totalmente automática por MITUI (em outras palavras, o desenvolvedor da aplicação não precisa escrever nenhum trecho de código para incluir essa funcionalidade na sua aplicação).

Cada elemento da barra de ações possui um caractere destacado, que permite ao usuário selecioná-lo diretamente (combinando esse caractere com a tecla *Alt*). Alternativamente, o usuário pode navegar na barra de ações com as teclas de cursor após pressionar a tecla *Alt* (ou a tecla *F10*) ou dar um clique de mouse sobre a opção que deseja ativar. As letras em destaque são modificadas para usar a mesma cor que o restante do nome dos comandos quando a barra de ações não pode ser utilizada.

A segunda escolha do usuário é processada através de menus pendentes, como os apresentados na Figura 5.7. Além das interações equivalentes às disponíveis na barra de ações, o projetista da aplicação pode definir *teclas aceleradoras* para permitir a usuários experientes acionar os

Figura 5.7 - Menus pendentes criados por MITUI

comandos do menu sem ter que fazer acesso a este. As opções dos menus pendentes podem estar inibidas (impedindo o usuário de escolhê-las) e/ou marcadas (o nome aparece precedido por uma marca $\checkmark$, indicando que o estado indicado pela opção está em uso no momento).

MITUI não permite apenas que a aplicação modifique o estado das opções de menu durante a execução, como também permite a inclusão e eliminação de opções durante a execução da aplicação. Desta maneira, as aplicações podem exibir menus contendo apenas aquelas opções que se aplicam a seu estado corrente.

5.4.2. Caixas de Diálogo

MITUI implementa caixas de diálogo modais: uma área retangular da tela da aplicação é sobreposta pelo diálogo, que objetiva obter ou fornecer informação do usuário. Enquanto o diálogo estiver ativo, o usuário não tem acesso a nenhum outro recurso da aplicação.

Para exibir informação em uma caixa de diálogo, a aplicação pode utilizar textos e molduras predefinidas e/ou áreas dinâmicas, cujo conteúdo muda durante a execução do diálogo sob controle da aplicação.

Para obter valores do usuário, MITUI fornece uma variedade de técnicas de interação predefinidas: caixas de verificação, múltipla escolha, listas, strings e botões de comando*. Veja a Figura 5.8 para ver alguns exemplos da apresentação visual de cada técnica de interação.

As caixas de verificação são adequadas para obter valores lógicos do usuário (verdadeiro/falso ou liga/desliga). Quando houver mais de duas possibilidades para o valor, é necessário utilizar a múltipla escolha, que permite ao usuário escolher um valor dentre um conjunto fixo de opções mutuamente exclusivas. Quando o conjunto de valores é muito grande ou pode mudar durante a execução do diálogo, torna-se necessário utilizar as listas, que podem ser maiores do que o espaço disponível na tela (nesse caso as opções serão deslocadas conforme o usuário for navegando na lista).

A entrada de strings arbitrários permite ao usuário digitar mais caracteres do que os que podem ser exibidos simultaneamente dentro do espaço disponível na caixa de diálogo. Cada entrada pode também ser associada a um predicado (uma função que é chamada com o valor atual do

---

* "Check boxes", "choice boxes" ou "radio buttons", "list boxes", "text boxes" e "command buttons", respectivamente, em inglês.

☒ Maiúsculas = Minúsculas
☒ Confirmar Cada Troca

Duas caixas de verificação

Componentes da tela:
☐ Barra de Ações
☐ Diálogos
☒ Janelas
☐ Barras de Deslocamento
☐ Fundo

Uma caixa de múltipla escolha

Arquivos em C:\DOS

4208.CPI	BISKCOMP.COM	EDIT.COM	GRAPHICS.COM
5202.CPI	BISKCOPY.COM	EDIT.HLP	GRAPHICS.PRO
ANSI.SYS	DISPLAY.SYS	EDLIN.EXE	HELP.EXE
APPEND.EXE	DOSHELP.BAK	EGA.CPI	JOIN.EXE
APPNOTES.TXT	DOSHELP.HLP	EGA.SYS	KEYB.COM
ASSIGN.COM	DOSKEY.COM	EXE2BIN.EXE	KEYBOARD.SYS
ATTRIB.EXE	DOSHELL.COM	EXPAND.EXE	LABEL.EXE
BACKUP.EXE	DOSHELL.EXE	FASTOPEN.EXE	LCD.CPI
CHKDSK.EXE	DOSHELL.GRB	FC.EXE	LOADFIX.COM
COMMAND.COM	DOSHELL.HLP	FDISK.EXE	MEM.EXE
COMP.EXE	DOSHELL.INI	FIND.EXE	MIRROR.COM
COUNTRY.SYS	DOSHELL.VID	FORMAT.COM	MODE.COM
DEBUG.EXE	DOSWAP.EXE	GORILLA.BAS	MONEY.BAS
	DRIVER.SYS	GRAFTABL.COM	MORE.COM

Uma caixa de lista contendo nomes de arquivos do MS-DOS 5.0

Procurar Por:
 MITUI.....
☐ Trocar Por:
 Um UIDS brasileiro.....

Digitação de strings

Um conjunto de quatro botões de comando

Figura 5.8 - As técnicas de interação para entrada de valores em caixas de diálogo

string e deve retornar verdadeiro ou falso para indicar se o valor é correto ou não), caso em que o diálogo só poderá ser confirmado se o string contiver um valor que satisfaz o predicado.

Finalmente, os botões de comando são usados para que o usuário indique o que deve acontecer com os dados exibidos e/ou digitados na caixa de diálogo. Tipicamente, existem botões de comando chamados "Ok" (usado para encerrar o diálogo confirmando os valores) e "Cancelar" (usado para anular a utilização dos valores). Ainda, a aplicação pode determinar se o diálogo deve ser encerrado quando o botão for selecionado pelo usuário e, caso exista, o nome de uma rotina da aplicação a ser chamada quando o usuário selecionar o botão. Desta forma, é possível construir inclusive diálogos aninhados, bastando que a rotina acionada através de um botão por sua vez inicie um novo diálogo com o usuário.

5.4.3. Janelas de Documento

Toda aplicação desenvolvida com MITUI pode usar janelas de documento para exibir ao usuário os resultados do trabalho já efetuado com a aplicação. Essas janelas são amodais, isto é, a sua presença não impede o acesso ao menu de comandos da aplicação.

Toda janela de documento criada com MITUI (veja a Figura 5.9) possui uma série de características predefinidas e dá ao usuário a possibilidade de efetuar uma certa quantidade de operações com ela.

Entre as características predefinidas das janelas de documento vale a pena citar a presença de um título para que o usuário identifique a janela, e o conteúdo, que é determinado pela aplicação. Caso os dados usados como conteúdo da janela sejam maiores que a própria janela, MITUI disponibiliza automaticamente barras de deslocamento para que o usuário possa saber qual parte dos dados ele está vendo.

```

C:\AUTOEXEC.BAT
@ECHO OFF
BREAK ON
VERIFY ON
PROMPT $p$g
PATH C:\XL;C:\WFW2;C:\WIN;C:\DOS;C:\WORD5;C:\
SET NU=C:\NU
SET TEMP=D:\
SET TMP=D:\
SMARTDRV A B C+ 2048 1280
LOADHIGH DOSKEY
LOADHIGH SHARE
NLSFUNC
MODE CON CP PREPARE=((437 850) \DOS\EGA.CPI)
MODE CON RATE=32 DELAY=1
LOADHIGH KEYB BR

```

Figura 5.9 - Uma janela de documento criada por MITUI

```

C:\AUTOEXEC.BAT
Restaurar
Proxima Ctrl+F6
Over Ctrl+F7
Amanho Ctrl+F8
Minimizar Ctrl+F9
Maximizar Ctrl+F10
Fechar Ctrl+F4
Subdividir
MODE CON CP PREPARE=((437 850) \DOS\EGA.CPI)
MODE CON RATE=32 DELAY=1
LOADHIGH KEYB BR

```

Figura 5.10 - Uma janela de documento criada por MITUI
com seu menu de controle aberto

As operações predefinidas incluem a possibilidade de o usuário modificar a qualquer momento o tamanho ou a posição da janela na tela. Como casos especiais desses, a janela pode ser maximizada para ocupar toda a região da tela disponível ou minimizada para ocupar o mínimo de espaço possível na tela. Também estão predefinidas operações para fechar as janelas e para permitir ao usuário subdividir o conteúdo da janela em vários painéis (o que permite ver em uma mesma janela várias regiões dos dados sendo manipulados). A Figura 5.10 exibe o menu de controle de uma janela de documento, usado principalmente por usuários que não dispõem de mouse para controlar esses atributos das janelas.

5.4.4. Barras de Deslocamento

No caso das janelas de documento e das caixas de lista e entradas de string das caixas de diálogo, MITUI permite que os dados existam em quantidade maior do que o espaço disponível na tela para a sua exibição.

Para orientar o usuário, MITUI usa barras de deslocamento como as da Figura 5.11. Cada barra de deslocamento possui um fundo pontilhado, um indicador de posição denominado *elevador* e duas setas nas extremidades. O elevador tem tamanho proporcional ao fundo pontilhado na mesma razão entre os dados visíveis dentro da janela e o total de dados disponíveis. Quando o elevador encosta numa extremidade, a seta dessa ponta é inibida (não pode mais ser objeto de um clique de mouse para forçar o deslocamento do conteúdo da janela).

5.4.5. Macros

Chama-se de *macros* o registro ("gravação") de uma seqüência de tokens de entrada gerada pelo usuário com o objetivo de simular ("reproduzir") esses mesmos tokens mais tarde automaticamente.

Como MITUI é um UIDS de geração automática, ele tem total controle sobre os tokens de entrada e saída. Este fato facilitou sobremaneira a implementação do recurso de macros para uso pelas aplicações. A aplicação pode comandar o início da gravação, a parada, a reprodução, a destruição e a leitura e gravação das macros em disco.

5.5. Programação com MITUI

Esta seção pretende ilustrar o uso do UIDS MITUI em aplicações. Obviamente, não é possível descrever todos os recursos com detalhes. Os exemplos apresentados são extraídos de [MBI 91], o Manual Tutorial distribuído junto com o software.

5.5.1. Uma Aplicação Vazia

Como primeiro exemplo vamos exibir uma aplicação vazia, que implementa como funcionalidade apenas a sua identificação para o usuário:

```
/* MEMPTY Programa MITUI vazio */  
  
#include <stdio.h>  
  
#include "\mitui\mt.h"  
  
void main(int argc, char *argv[]);
```

```
void main(int argc, char *argv[]) {
    ApplicationStart(argv[0], NULL);
}

void About(boolean PRIMEIRA_VEZ) {
    dialogBox *SOBRE = DialogBoxCreate("Programa vazio",
                                       10L, 26L, 4L, 26L);
    DialogBoxText(SOBRE, 0L, 1L, "Programa Demo MITUI vazio",
                 DialogBoxColor(DialogText, Current));
    DialogBoxButton(SOBRE, 1L, 10L, "&Ok", NULL, TRUE);
    DialogBoxRun(&SOBRE);
}
```

Embora este programa seja bastante curto, ele é uma aplicação MITUI completa. Todo módulo de programa de uma aplicação MITUI deve incluir o arquivo de cabeçalho* de nome `mt.h`. Ele contém as definições de todos os tipos usados por MITUI e de todas as funções que compõem a API** de MITUI.

A inclusão do arquivo de cabeçalho `stdio.h`, fornecido junto com os compiladores de linguagem C, é necessária para poder usar `NULL` como valor de ponteiro. Uma aplicação MITUI pode incluir arquivos de cabeçalho da biblioteca padrão, como é o caso aqui, ou de qualquer outra biblioteca que desejar.

Como todo programa C, a aplicação deve ter uma função de nome `main`. As aplicações MITUI têm obrigatoriamente pelo menos mais uma função, que deve ter o nome `About` e serve para identificar a aplicação para o seu usuário.

---

* "Header file", em inglês.

** Abreviatura de "Application Program Interface" em inglês, ou Interface dos Programas Aplicativos. É a denominação dada ao conjunto de procedimentos disponibilizados por um componente de software básico para a construção de software aplicativo.

A função `main` de um aplicativo MITUI precisa fazer o acesso aos argumentos da linha de comando, mas não retorna o controle ao sistema operacional (essa é uma das tarefas de MITUI), de sorte que o tipo do valor de retorno de `main` deve ser declarado como sendo do tipo `void` (vazio, em C).

Dentro do bloco de comandos da função `main`, existe apenas um único comando: a chamada da função `ApplicationStart`, que é a primeira função da API de MITUI sendo utilizada. Quando ela é chamada, ela assume o controle da execução e nunca mais retorna para a aplicação que a chamou. Ao invés disso, MITUI executa seu código de inicialização e inicia a repetição de tratamento dos tokens de entrada gerados pelo usuário. Se o usuário solicitar o encerramento da aplicação, então `ApplicationStart` retorna o controle diretamente ao sistema operacional.

`ApplicationStart` possui dois argumentos. O primeiro deles é o nome do arquivo executável correspondente à aplicação, tal como passado à aplicação pelo sistema operacional. MITUI usa este nome para automaticamente procurar (e/ou salvar, se for o caso) os arquivos de configuração e de ajuda específica da aplicação, que usam o mesmo nome do arquivo executável, mas com a extensão trocada para `.CFG` e `.HLP`. O segundo argumento vale `NULL` em aplicações simples.

A função `About` é chamada por MITUI em duas situações: ela é chamada automaticamente de dentro de `ApplicationStart` logo após a execução do código de inicialização de MITUI para informar o usuário qual a aplicação que ele está iniciando. A outra chamada de `About` ocorre quando o usuário escolhe a opção `Acerca De...` do menu pendente `Ajuda`.

Para que `About` saiba se está sendo chamada pela primeira vez (esse é um instante conveniente para executar código de inicialização da aplicação que requeira que a inicialização de MITUI já tenha sido executada), ela recebe um argumento do tipo `boolean` que tem o valor `TRUE` se e somente se está sendo chamada no início da aplicação. Por exemplo, se a aplicação trata os arquivos indicados pelo usuário na linha de comando através da sua

exibição em janelas, **About** pode criar essas janelas antes que o usuário inicie sua atividade com a aplicação.

A função **About** do exemplo utiliza uma caixa de diálogo para identificar a aplicação. Ela inicia chamando a função **DialogBoxCreate** da API de MITUI, que cria uma área de controle para uma nova caixa de diálogo vazia. A seguir ela chama as funções **DialogBoxText** e **DialogBoxButton** para adicionar uma mensagem e um botão de comando à caixa de diálogo respectivamente. Finalmente, a caixa de diálogo é exibida, o usuário interage com ela e ela é eliminada da memória através da chamada da função **DialogBoxRun**. Analisaremos cada uma dessas quatro funções com mais detalhes.

DialogBoxCreate retorna à aplicação o endereço da área de controle da caixa de diálogo, que deve ser armazenado (no ponteiro **SOBRE** no nosso exemplo acima) para poder continuar a trabalhar com a caixa de diálogo. Os argumentos de **DialogBoxCreate** são o título da caixa de diálogo, as coordenadas de tela do seu canto superior esquerdo e a altura e a largura da mesma.

DialogBoxText adiciona seu argumento tipo string à caixa de diálogo. Note o uso do ponteiro retornado por **DialogBoxCreate** para identificar o diálogo. Os demais argumentos são as coordenadas do texto dentro da caixa de diálogo e a cor a ser usada para exibir o texto. No exemplo, é usada a cor correntemente definida pelo usuário como a cor para o texto das caixas de diálogo, que é obtida chamando a função **DialogBoxColor** com os argumentos **DialogText** e **Current**.

DialogBoxButton adiciona um botão de comando à caixa de diálogo. Esta função tem três argumentos, além do ponteiro da caixa de diálogo e das coordenadas do canto superior esquerdo da área a ser ocupada pelo botão. O primeiro argumento adicional é a legenda do botão ("**&ok**" no exemplo; o símbolo **&** precede o caractere destacado), o segundo argumento extra é o endereço de uma função a ser chamada quando o botão for pressionado (**NULL** no exemplo, já que nenhuma função precisa ser chamada) e o último

argumento indica se o diálogo deve ou não (**TRUE** no exemplo) ser encerrado depois que o botão tenha sido pressionado pelo usuário.

Finalmente, a caixa de diálogo é executada através da chamada da função **DialogBoxRun**. Ela tem como único argumento o endereço onde a aplicação armazenou o ponteiro da caixa de diálogo (para que possa anular o ponteiro depois que o usuário encerre o diálogo e a memória associada já tenha sido liberada) e retorna o número do botão pressionado pelo usuário para encerrar o diálogo (um para o primeiro, dois para o segundo, etc.; um valor zero ou negativo indica o cancelamento do diálogo). Como a caixa de diálogo do exemplo acima possui apenas um botão de comando e também não há entrada de dados, o valor de retorno da função é ignorado.

Para compilar uma aplicação **MITUI**, basta compilar o(s) módulo(s) da aplicação normalmente* e incluir a biblioteca **MITUI**, um arquivo de nome **mt.lib** no processo de link-edição.

5.5.2. Uma Aplicação com Menus e Janelas

Como segundo exemplo descreveremos uma aplicação do tipo **Alô mundo**, capaz de lidar com múltiplas janelas e que usa um menu com opções que são habilitadas e inibidas durante a execução.

```
/*  MTHELLO      Programa MITUI Alô mundo
    com opções de menu pendente habilitadas e inibidas */

#include <stdio.h>

#include "\mitui\mt.h"
```

---

* **MITUI** está disponível em versões para os compiladores de linguagem C e C++ da Borland e da Microsoft. As únicas opções de compilação obrigatórias são o uso do modelo de memória **large** (ou **huge**) e do tipo **unsigned char** para as variáveis declaradas como **char**.

```
void AloMundoAbrir (void),
    AloMundoFechar(void),
    AloMundoSalvar (window *JANELA);
char *AloMundoConteudo(
    window *JANELA, long Y, long X, long COMPRIM);
void main(int argc, char **argv);

void main(int argc, char **argv) {
    ActionBarInsert("A&lô", "Abrir ou fechar uma "
        "janela exibindo \"Alô mundo!\", 1, NULL);
    PullDownInsert("Alô", "&Abrir",
        "Abrir uma janela exibindo \"Alô mundo!\",
        1, F3, AloMundoAbrir);
    PullDownInsert("Alô", "&Fechar",
        "Fechar uma janela exibindo \"Alô mundo!\",
        2, F4, AloMundoFechar);
    PullDownChange("Alô", "Fechar", DISABLE);
    ApplicationStart(argv[0], NULL);
}

void About(boolean PRIMEIRA_VEZ) {
    color TEXT = DialogBoxColor(DialogText, Current);
    dialogBox *SOBRE = DialogBoxCreate(
        "Sobre Alô mundo!", 10L, 26L, 5L, 28L);
    DialogBoxText(SOBRE, 0L, 1L,
        "Programa MITUI Alô mundo", TEXT);
    DialogBoxButton(SOBRE, 2L, 11L, "&Ok", NULL, TRUE);
    DialogBoxRun(&SOBRE);
}

static long QTD_JANELAS = 0;

void AloMundoAbrir() {
    WindowRedraw(WindowCreate("Alô",
        QTD_JANELAS, 3 * QTD_JANELAS, 2L, 15L, 1L, 10L,
        AloMundoConteudo, NULL, AloMundoSalvar));
    if (++QTD_JANELAS == 20)
        PullDownChange("Alô", "Abrir", DISABLE);
    else if (QTD_JANELAS == 1)
        PullDownChange("Alô", "Fechar", NORMAL );
}
```

```
}

void AloMundoFechar() {
    window *JANELA = WindowInFocus(NULL);

    WindowClose(JANELA);
    WindowDestroy(&JANELA);
    WindowRedrawAll();
}

void AloMundoSalvar(window *JANELA) {
    if (--QTD_JANELAS == 19)
        PullDownChange("Alô", "Abrir" , NORMAL );
    else if (QTD_JANELAS == 0)
        PullDownChange("Alô", "Fechar", DISABLE);
}

char *AloMundoConteudo(
    window *JANELA, long Y, long X, long COMPRIM
) {
    return (Y == 0 ? "Alô mundo!" : "          ") + (int) X;
}
```

Esta aplicação exibe até vinte janelas contendo a mensagem **Alô mundo!** ao mesmo tempo. O usuário determina a abertura e fechamento das janelas através das opções **Abrir** e **Fechar** do menu pendente **Alô**. Todas as janelas podem ser movimentadas, redimensionadas, maximizadas, minimizadas, fechadas e subdivididas (horizontal e verticalmente) em painéis sem a necessidade de qualquer trecho de código para esse fim dentro da aplicação.

A função **ActionBarInsert**, chamada dentro de **main** antes da chamada de **ApplicationStart**, inclui um novo comando na barra de ações. Seus argumentos são o nome do comando, a mensagem associada (exibida na linha de status sempre que o usuário colocar o cursor sobre o nome do comando), a sua posição na sequência de comandos e o nome da função a ser

acionada caso o comando seja escolhido (`NULL` no exemplo indica que não há função a ser chamada).

A função `PullDownInsert` deve ser chamada para criar as opções de menu pendente. Ela possui seis argumentos: o nome do comando da barra de ações ao qual a opção sendo criada será subordinada, o nome da opção, a mensagem associada, a posição dentro do menu pendente, o nome da tecla aceleradora (`Null` se não houver) e o nome da função a ser chamada caso quando a opção for escolhida (`AloMundoAbrir` e `AloMundoFechar`).

Todas as funções acionadas pelo menu devem ser funções sem argumentos e sem valor de retorno, como é o caso de `AloMundoAbrir` e `AloMundoFechar`.

A implementação de `AloMundoAbrir` chama a função `WindowRedraw` usando como argumento o valor retornado pela função `WindowCreate`, que é o endereço da área de controle criada por `MITUI` para a janela.

`WindowCreate` possui muitos argumentos, mas a maioria são simples: o primeiro argumento é o título inicial da janela, depois seguem as coordenadas do canto superior esquerdo da janela, a altura e a largura da janela na tela e a altura e largura *virtuais* iniciais. Estes dois últimos (1 e 10 no exemplo) são a altura e a largura total dos dados que o usuário poderá visualizar através da janela.

A *área virtual de dados* é um dos conceitos mais importantes usados por `MITUI`. Cada janela exibe permanentemente alguma(s) parte(s) da sua área virtual de dados. A aplicação não tem o controle sobre qual parte dos dados o usuário visualiza através da janela a cada instante, já que o usuário pode modificar o tamanho da janela, deslocar os dados visíveis dentro da janela ou até mesmo subdividir a janela sem nenhum tratamento desses recursos dentro da aplicação.

No entanto, mesmo controlando todos esses eventos automaticamente, `MITUI` precisa que a aplicação forneça os dados a serem exibidos dentro das janelas. A totalidade desses dados é de responsabilidade da aplicação. No

exemplo acima, os dados ocupam apenas uma linha de dez caracteres contendo a mensagem `Alô mundo!`.

MITUI se encarrega de redesenhar as janelas abertas de forma independente da aplicação toda vez que o usuário efetuar uma modificação de tamanho, posição, subdivisão ou outro atributo da janela. No entanto, **MITUI** precisa obter da aplicação quais os dados a serem exibidos. Para esse fim, a aplicação deve fornecer uma *função de conteúdo*, que informa a **MITUI** sobre o conteúdo da área virtual de dados. No exemplo, essa função chama `AloMundoConteudo`.

Toda função de conteúdo é chamada por **MITUI**, toda vez que precisar usar uma parte da área virtual de dados, com quatro argumentos: o ponteiro da janela (definido quando a janela foi criada), as coordenadas iniciais virtuais dos dados (o número da linha e da coluna dentro do total dos dados da aplicação) e o comprimento dos dados sendo solicitados. A função de conteúdo deve colocar esses dados em algum lugar na memória, se não estiverem já carregados, e retornar o endereço inicial deles.

Normalmente cada janela criada pela aplicação possui um conteúdo diferente. Como este não é o caso no exemplo, a mesma função de conteúdo serve para todas as janelas. Conseqüentemente, não há a necessidade de a aplicação armazenar os valores dos ponteiros de janela retornados por `WindowCreate`.

Note que os dados não precisam ser obrigatoriamente armazenados na memória durante todo o tempo em que a aplicação está executando: aplicações mais complexas do que a do exemplo anterior podem armazenar seus dados (eventualmente de forma parcial) em arquivos em disco, e trazê-los para a memória somente quando forem solicitados por **MITUI** à função de conteúdo.

Os dois últimos argumentos de `WindowCreate` também são funções, que podem ser omitidas opcionalmente (indicando o valor `NULL`). A primeira das funções (que vale `NULL` no exemplo) é uma função a ser acionada por

MITUI toda vez que o usuário gerar um token de entrada sobre a janela que não possua nenhum significado predefinido nas técnicas de interação do padrão CUA. Desta forma, a aplicação pode definir funcionalidade adicional para as janelas.

A última função argumento de **windowCreate** é a função a ser acionada por **MITUI** quando a janela for encerrada pelo usuário. Esta função é responsável por salvar o conteúdo da janela em um arquivo em disco e/ou executar qualquer outro código de "desinicialização" da janela.

No exemplo, a função de encerramento é **AloMundoSalvar**. Ela é usada para manter o valor da variável **QTD_JANELAS**, que é incrementada na função **AloMundoAbrir** (quando a janela é criada). Essa variável sempre indica o número total de janelas abertas no momento.

Quando o usuário usar a opção **Alt F**, é acionada a função **AloMundoFechar**. Ela chama primeiramente a função **windowInFocus** para obter o ponteiro da janela ativa no momento (que é a que deve ser fechada). Mesmo que a aplicação tivesse armazenado os ponteiros das janelas quando estas foram criadas, ainda assim teria que chamar **windowInFocus**, já que o usuário pode escolher qualquer janela para ser a ativa sem a intervenção da aplicação. Para fechar a janela, basta chamar as funções **windowClose** (que indiretamente aciona a função de encerramento da janela) e **windowDestroy**, que elimina a definição da janela da memória. Finalmente, **AloMundoFechar** chama a função **windowRedrawAll** para atualizar a imagem das janelas na tela, já que a janela fechada pode ter estado na frente de (parte de) outras janelas.

Note que as funções **AloMundoAbrir** e **AloMundoFechar** não contêm verificações para evitar a criação de janelas além do permitido ou tentar fechar janelas quando já não há mais janelas abertas. A melhor maneira de evitar esse tipo de erros do usuário consiste em preveni-los: a aplicação deve tornar as opções **Abrir** e **Fechar** disponíveis apenas nas situações corretas.

Para inibir opções não aplicáveis de um menu pendente, MITUI fornece a função `PullDownChange`. Ela deve ser chamada com três argumentos: o nome do comando da barra de ações e o nome da opção do menu pendente correspondente, que terá seu estado alterado, e o novo estado da opção (que pode ser `NORMAL` ou `DISABLE` para respectivamente habilitá-la e inibi-la).

Note como no exemplo a função `AlomundoAbrir` contém uma chamada a `PullDownChange` que desativa a ela própria ao criar a vigésima janela. Ela habilita a opção `Fechar` quando é aberta a primeira janela (já que essa opção não está habilitada no estado inicial da aplicação).

Analogamente, a função `AlomundoSalvar` habilita a opção `Abrir` se ela fecha a vigésima janela e inibe a opção `Fechar` quando fecha a última janela ainda aberta. A variável `QTD_JANELAS` é mantida exatamente para esse fim. Como ela é decrementada na função de encerramento de janela, essa variável mantém o valor correto mesmo que alguma(s) janela(s) seja(m) fechada(s) pelo usuário através do menu pendente de controle da janela (ao invés de usar a opção `Fechar` do menu pendente `Alô`).

Apesar do reduzido tamanho da aplicação do exemplo acima, ela já inclui todas as características básicas do modelo de desenvolvimento de aplicações utilizado por MITUI.

5.6. A Implementação de MITUI

Nesta seção descreveremos as principais características da implementação de MITUI. O próprio código fonte de MITUI foi escrito totalmente em linguagem C. Ele é composto de uma quantidade bastante grande de módulos diferentes (veja a Figura 5.11 para ter um resumo dos módulos).

MTACTION.C	Módulo de tratamento da barra de ações
MTCONFIG.C	Módulo de tratamento da configuração
MTCONTRL.C	Módulo de tratamento do menu de controle
MTDLGBOX.C	Módulo de tratamento de caixas de diálogo
MTEVENTS.C	Módulo de tratamento de erros
MTEVENTS.C	Módulo principal, repetição sobre eventos
MTHelp.C	Módulo de implementação do sistema de ajuda
MTKEYBRD.C	Módulo de detecção de tokens no teclado
MTMACROS.C	Módulo de implementação das macros
MTMOUSE.C	Módulo de detecção de tokens de mouse
MTOTHERS.C	Módulo contendo outras funções
MTSCREEN.C	Módulo contendo funções de tela
MTSCROLL.C	Módulo de tratamento de barras de deslocamento
MTSTRING.C	Módulo contendo variáveis dependentes de idioma
MTVIDEO.C	Módulo de acesso ao sistema de vídeo
MTWINCTL.C	Módulo de tratamento de eventos no menu de controle de janelas
MTWINDOW.C	Módulo de gerenciamento de janelas
MTWINRUN.C	Módulo de tratamento de eventos de janela
MTHLPMKR.C	Código fonte de MITUI HelpMaker (compilador de textos para o sistema de ajuda)

Figura 5.11 - Os módulos fonte que compõem a biblioteca MITUI

O módulo de mais alto nível é `MTEVENTS.C`: é ele que contém a função `ApplicationStart`, pela qual o aplicativo passa o controle para MITUI. Seu pseudocódigo é:

```

Registrar tratamento de erros de hardware
Registrar tratamento de interrupção da aplicação
Armazenar disco e diretório originais
Detectar tipo de placa de vídeo em uso
Inicializar o sistema de vídeo
Inicializar o sistema de mouse

```

```
Carregar configuração de vídeo e mouse da aplicação
Inicializar o sistema de ajuda
Registrar função de encerramento da aplicação
Desenhar a barra de ações inicial
Inicializar a área de janelas
Chamar a função About de identificação da aplicação
para sempre {
    exibir coordenadas virtuais do cursor na janela ativa
    obter o próximo evento
    se não estiver executando macro
    e foi solicitada execução de macro já gravada {
        Ativar execução da macro
        Definir como evento corrente o primeiro da macro
    }
    se evento é tecla de acesso ao menu
    ou mouse foi clicado na área do menu na tela
        acionar tratador de eventos de menu
    senão
        acionar tratador de eventos de janela
}
```

`ApplicationStart` inicializa a aplicação e depois executa uma repetição incondicional, que será interrompida quando o evento de menu for o encerramento da aplicação. Nesse momento, é acionada a função de encerramento, que não faz parte das funções documentadas como estando disponíveis para a aplicação. A aplicação aciona esta função indiretamente, através de uma chamada à função `exit` da biblioteca padrão de funções (definida pelo padrão ANSI da linguagem C [PLAUGER 90]). Seu pseudocódigo é:

```
marcar início do encerramento da aplicação
registrar tratamento especial de interrupção da aplicação
se a aplicação registrou função de encerramento
    acionar função de encerramento da aplicação
para cada janela ativa
    acionar função de encerramento da janela
se a configuração atual não estiver salva em disco
    se a memória não está esgotada {
```

```
        perguntar ao usuário se deve salvar configuração
        se ele confirmar
            salvar a configuração em disco
    }
    senão
        salvar a configuração em disco
    desinicializar o sistema de ajuda
    desinicializar o sistema de vídeo
    restabelecer o disco e diretório originais
    devolver o controle para o sistema operacional
```

O papel mais importante do módulo **MTEVENTS.C** é a obtenção dos tokens de entrada e o acionamento do tratamento correspondente ao evento gerado pelo usuário. A obtenção dos tokens de entrada é uma função separada, que chama os módulos de teclado e mouse (ou o módulo de macros, caso esteja sendo reproduzida uma), já que os módulos de tratamento de eventos podem precisar obter tokens de entrada adicionais àquele que determinou a sua ativação.

5.6.1. Módulos de baixo nível

Para facilitar o porte futuro de **MITUI** para outros ambientes de execução (diferentes de PCs rodando o sistema operacional **MS-DOS**), todo o acesso aos dispositivos de entrada e saída está encapsulado em módulos separados.

Os módulos de acesso aos dispositivos de entrada são os módulos de detecção e obtenção de tokens de entrada do teclado, **MTKEYBRD.C**, e do mouse, **MTMOUSE.C**. O módulo de acesso ao dispositivo de saída, o terminal de vídeo, chama-se **MTVIDEO.C**.

No caso de um porte para um ambiente como o modo texto do **OS/2**, apenas esses módulos deverão sofrer alterações.

O módulo **MTKEYBRD.C** utiliza várias tabelas com valores constantes predefinidos para determinar o código do token de entrada a partir da tecla

física apertada pelo usuário e o estado das teclas de status *Shift*, *Ctrl*, *Alt*, *Num Lock*, *Caps Lock* e *Scroll Lock*. MITUI utiliza um sistema próprio de codificação dos tokens de entrada, definido no arquivo de cabeçalho `MT.H` pelo tipo `event`, que é um superconjunto do conjunto ASCII estendido definido pela IBM para uso em PCs.

Além de disponibilizar o código e o nome dos tokens, o módulo `MTKEYBRD.C` disponibiliza uma rotina que informa se há uma tecla disponível e outra que informa qual a tecla disponível.

O único aspecto incomum na implementação deste módulo diz respeito ao tratamento da tecla *Alt*. Embora esta tecla não seja reconhecida pelo hardware como uma tecla que gere um código, o padrão CUA a define como um dos métodos de acesso à barra de ações. MITUI define *Alt* como um dos valores do tipo `event`.

A lógica envolvida no uso da tecla *Alt*, no entanto, é um tanto quanto complicada: basicamente é necessário lembrar do fato que a tecla foi pressionada, para gerar o token de entrada quando a tecla for solta. No entanto, se uma outra tecla (ou um dos botões do mouse) for pressionada simultaneamente com *Alt* (como por exemplo para gerar o token de entrada *Alt+A* ou *Alt+Click*), então **não** deve ser gerado o token *Alt* quando a tecla for solta.

Toda a informação necessária para esse fim é armazenada em variáveis estáticas do módulo, já que a rotina de verificação da presença de um token de entrada no teclado sempre deve retornar imediatamente (se ela ficasse aguardando o próximo token, o restante da aplicação pareceria suspenso, impedindo, por exemplo, a execução de tarefas em segundo plano*). Essas variáveis são consultadas a cada chamada para determinar como a rotina deve proceder.

---

* "Background execution", em inglês.

Essas mesmas variáveis servem de base à rotina que fornece o token de entrada para a aplicação.

No caso do mouse, a quantidade de tokens de entrada é menor, mas a lógica das rotinas de detecção e obtenção de valores do tipo `event` são ainda mais complicadas, porque no caso do mouse existem ainda mais dependências temporais: tanto o arrasto como o clique duplo não podem ser gerados como tokens de entrada como resposta imediata a uma ação do usuário. No entanto, a aplicação deve realimentar cada token do usuário imediatamente.

Para resolver este conflito, o módulo `MTMOUSE.C` usa a seguinte tática caso o botão esquerdo do mouse seja pressionado:

- se o botão esquerdo do mouse permanecer apertado por um período de tempo significativo, sem que o mouse seja movimentado, é gerado o token de entrada `LeftClick`, combinado com as teclas de status ativas ao iniciar a operação do mouse;
- se o mouse for movimentado com o botão pressionado, então é gerado o token `Drag` (correspondente ao arrasto), combinado com as teclas de status ativas ao iniciar a operação do mouse;
- se o botão do mouse já foi solto, e o mouse tiver sido movimentado ou tiver passado um período de tempo significativo, é gerado o token de entrada `LeftClick`, caso contrário existe a possibilidade de gerar o token de entrada `DoubleClick`, cada um combinado com as teclas de status ativas ao iniciar a operação do mouse.

A quantidade exata de tempo considerada como um período significativo foi definida como sendo inversamente proporcional à sensibilidade do mouse, que o usuário pode escolher através de uma das opções do menu de controle da aplicação. Quanto mais sensível for a configuração corrente do mouse, tanto menos tempo será considerado para verificar a possibilidade de um clique duplo.

O módulo **MTMOUSE.C** também implementa os demais recursos de mouse necessários, como por exemplo a sua inicialização (que inclui a detecção da presença ou ausência do mouse), a configuração da sua sensibilidade e da cor do ponteiro do mouse na tela, a exibição e inibição do ponteiro (já que ele não pode estar sendo exibido enquanto a tela é atualizada para evitar de "desaparecer" temporariamente sem explicação aparente para o usuário), a definição da área da tela onde ele pode ser usado (o mouse não pode ser usado na área de status da aplicação; no caso de uso das caixas de diálogo, o mouse também só pode ser usado dentro da região da tela ocupada por estas), funções para salvar e restaurar o estado corrente do mouse, entre outras.

O último módulo de baixo nível é **MTVIDEO.C**, responsável pela interação da aplicação com a placa de vídeo (e o monitor). Dentre outros recursos implementados, vale a pena citar a inicialização e "desinicialização" do sistema de vídeo, a detecção do tipo de placa de vídeo em uso (da qual dependem vários recursos, como o uso de 43 linhas na tela, ao invés das 25 normais, com placas EGA e até 50 linhas com placas VGA), as rotinas de saída de caracteres, cores ou ambos simultaneamente em uma determinada região da tela (fazendo acesso diretamente a memória de vídeo das placas para obter um desempenho satisfatório), as rotinas para salvar e restaurar o conteúdo (de parte) do vídeo (necessário quando uma região da tela é sobreposta temporariamente, como no caso da abertura de um menu pendente ou de uma caixa de diálogo) e as funções de manipulação do cursor no vídeo, como o posicionamento, a sua inibição e reexibição e a sua mudança de tamanho, entre muitas outras.

Este módulo é, dentre todos que compõem o código fonte de **MITUI**, o que possui a maior dependência das características do equipamento específico sendo utilizado para a execução da aplicação pelo usuário.

5.6.2. Módulos de tratamento de eventos

A partir da detecção dos tokens de entrada pela repetição contida em **MTEVENTS.C**, são acionados os demais módulos de tratamento de eventos.

O primeiro tratador de eventos citado acima corresponde aos eventos de menu. Esse tratador é implementado pelo módulo **MTACTION.C**, que inclui as funções da API de **MITUI** relativas ao menu e chamadas pela aplicação, além do tratador de eventos. A implementação das funções da API é baseada em uma estrutura de dados que mantém todas as variáveis necessárias à implementação do menu.

Já a rotina que implementa o tratador de eventos de menu é baseada em um autômato finito de estados, que permite diferenciar o uso do menu fechado do uso da barra de ações sozinha e do uso da barra de ações com menus pendentes já abertos. Essa distinção é necessária, já que o mesmo token de entrada muda de significado, conforme o estado corrente do menu.

Ainda, **MTACTION.C** inclui algumas rotinas auxiliares para determinar, por exemplo, se um token de entrada específico corresponde a uma das teclas aceleradoras definidas pela aplicação, ou se um clique de mouse foi dado sobre uma opção ativa de um menu pendente.

O tratamento de eventos no menu é baseado em uma repetição que responde aos tokens de entrada do usuário até que ele cancele o uso do menu ou acione a execução de uma rotina da aplicação. Enquanto no primeiro caso o controle retorna ao tratador principal de eventos imediatamente, isso só ocorre no segundo caso após a execução da rotina da aplicação.

Além de responder ao usuário através de rotinas associadas às opções do menu, a aplicação pode executar código quando o usuário gera tokens de entrada sobre janelas. O tratamento das janelas foi dividido, devido a seu tamanho, em três módulos diferentes: **MTWINDOW.C**, **MTWINRUN.C** e **MTWINCTL.C**.

Desses três módulos, o primeiro implementa as funções da API que manipulam janelas, modificando os valores da estrutura interna de dados correspondente. O segundo contém o tratador de eventos de janela propriamente dito, enquanto o último módulo citado contém o tratador de eventos gerados sobre o menu de controle das janelas.

O tratamento de eventos de janela não contém repetições. São implementados todos os eventos determinados pelo padrão CUA para a movimentação do cursor, da janela e dos dados, para a seleção de trechos de dados (inclusive a seleção simultânea de múltiplos trechos), e todos os eventos de controle da própria janela, como a mudança de posição e tamanho, sua maximização e minimização, sua subdivisão em (até quatro) painéis e o seu encerramento.

O módulo **MTDLGBOX.C** implementa as funções da API para caixas de diálogo e o respectivo tratador de eventos. Como as caixas de diálogo implementadas por **MITUI** são modais, o tratador de eventos estabelece a sua própria repetição de tratamento de eventos, que é encerrada junto com o encerramento da caixa de diálogo.

O módulo **MTMACROS.C** implementa as funções da API para a utilização de macros de eventos. As macros são armazenadas em variáveis locais ao módulo. As rotinas de verificação de disponibilidade de um token de teclado ou mouse consultam este módulo para verificar se há uma macro sendo reproduzida. Se for o caso, então há um token de entrada disponível para a aplicação (o próximo da macro). Analogamente, as rotinas de entrada dos tokens informam o módulo de macros sobre o token caso a gravação de macros tenha sido acionada.

O módulo **MTSCROLL.C** implementa as barras de deslocamento, tanto do ponto de vista de seu armazenamento como do tratamento de eventos. No entanto, este módulo não disponibiliza nenhuma de suas rotinas à aplicação, exceto a rotina de configuração de cores. Todo o restante do tratamento de barras de deslocamento (seja em entradas de string ou caixas de lista no caso

de diálogos, ou junto com janelas) é totalmente automático: para a aplicação, ele se passa como se não existisse.

5.6.3. Outros módulos fonte

Todos os recursos acessíveis ao usuário da aplicação através do menu de controle da aplicação são implementados nos módulos **MTCTRL.C** e **MTCONFIG.C**. Este último contém o código de leitura e gravação do arquivo de configuração da aplicação, enquanto que todos os outros recursos são implementados no primeiro.

As opções de menu do sistema de ajuda são implementadas no módulo **MTHELP.C** como um conjunto de caixas de diálogo que exibem as informações contidas nos arquivos de ajuda.

Os três módulos acima são módulos no mesmo nível dos módulos da aplicação. Eles são implementados usando exclusivamente os recursos de **MITUI** disponíveis também para a aplicação.

A necessidade de dispor de versões do código de **MITUI** contendo as mensagens em inglês ou em português resultou na criação do módulo **MTSTRING.C**. Este módulo utiliza compilação condicional para escolher os valores adequados para todas as variáveis que dependem da língua sendo usada, o que inclui todas as mensagens apresentadas na tela ao usuário por **MITUI** e variáveis relacionadas, como por exemplo o comprimento dessas mensagens.

O único outro módulo da implementação de **MITUI** que depende da língua alvo é o módulo de tratamento de erros de execução. Este módulo implementa não apenas o tratamento de erros críticos (normalmente tratados pelo MS-DOS), mas também os erros de execução gerados internamente na biblioteca como reação a um erro de codificação na aplicação ou outra situação anormal. Todos os erros de execução são codificados dentro da biblioteca, tornando os demais módulos independentes das mensagens. Para

facilitar o trabalho do desenvolvedor, os erros são detalhados também na documentação impressa de MITUI.

5.7. Uma avaliação da implementação

Como pode ser deduzido da seção anterior, a implementação de MITUI foi um projeto altamente não trivial. O total de código fonte supera o megabyte. Comparativamente, o código tem tamanho equivalente a quatro a cinco vezes o de um compilador sem otimização para a linguagem C ANSI completa que seja desenvolvido usando grafos sintáticos simples, como descritos em [SETZER 89], e que gere saída em Assembler.

O projeto lógico original de MITUI foi elaborado usando o modelo de eventos e o paradigma da orientação a objetos. No entanto, a obediência ao padrão CUA determinou que a hierarquia de classes necessária fosse totalmente estática. A criação e destruição de tratadores de eventos, como descrita na formalização desse modelo no Capítulo 3, também resultou totalmente previsível. A implementação não só tirou proveito desses fatores para obter uma implementação mais eficiente, como foram exatamente esses fatores que tornaram possível a implementação em uma linguagem tradicional como C, sem orientação a objetos*.

Quanto à comunicação entre o código de MITUI e o código da aplicação, MITUI utiliza o controle externo. A aplicação torna disponível procedimentos e funções a serem chamados, quando necessário, pelo código de MITUI. No entanto, esses procedimentos e funções podem conter, por sua vez, chamadas a rotinas fornecidas por MITUI.

---

* O uso de C como linguagem de implementação (ao invés de, por exemplo, C++) também é consequência de considerações a respeito do tamanho do mercado potencial para o produto.

A utilização do controle externo com procedimentos e funções chamados na aplicação permite separar o código de interface da semântica da aplicação sem perder eficiência no código total resultante para a aplicação.

Ainda, o controle externo permite ao MITUI fornecer a execução de tarefas em segundo plano, mesmo em ambientes onde o sistema operacional não o permita. Para tanto, a aplicação deve indicar um procedimento a ser chamado pelo MITUI toda vez que detectar a inexistência de qualquer evento a ser processado. Outra característica do MITUI cuja implementação foi facilitada pelo controle externo são as macros de eventos.

Durante o desenvolvimento do MITUI, foi necessário ter uma aplicação que explorasse todos os recursos da ferramenta. Para este fim foi criado um editor de textos simples, sem formatação, chamado **MITUI Editor**, mas que explora todos os recursos de uma interface padrão CUA. Este editor foi colocado no domínio público, com seu código fonte. A Figura 5.12 exibe a sua identificação inicial.

Este editor é uma aplicação útil e real, de tamanho médio. Seu menu possui aproximadamente 40 comandos diferentes; a sua implementação usando MITUI precisou de aproximadamente 5000 linhas de código fonte.

Considerando o código fonte da primeira versão* de MITUI junto com o código específico do editor, verificamos que este último corresponde a apenas 17% do total. Em relação ao tempo que foi gasto no desenvolvimento, apenas 14% corresponde ao código específico do editor.

---

* O uso do código fonte da primeira versão permite obter valores mais realistas, porque a principal mudança do código fonte nas versões seguintes, quanto ao seu tamanho, corresponde à possibilidade de gerar versões do software em inglês e português. Assim, todas as strings de caracteres e variáveis relacionadas com a exibição ocorrem pelo menos duas vezes (nos módulos fonte **MTSTRING.C** e **MTERRORS.C** e em todo o código fonte do editor), em linhas separadas, junto com os comandos de compilação condicional necessários. Acreditamos que esta situação não seja representativa de uma aplicação típica.

Figura 5.12 - A identificação de MITUI Editor

Podemos concluir que a utilização de **MITUI** trouxe, portanto, uma economia no esforço total de desenvolvimento do editor correspondente a um fator de cinco a seis vezes. Note que este valor é compatível com os resultados, já citados, obtidos com a utilização do **UIDS MacApp** [SCHMUCK 86].

Os comentários seguintes, extraídos de [OLSEN 86] foram escritos em relação ao **UIDS** chamado **MIKE**, já citado no Capítulo 4. No entanto, essas observações aplicam-se também a **MITUI**:

- "Em relação à saída, nada de novo foi utilizado em relação à independência dos dispositivos físicos; no entanto, há uma contribuição significativa em relação à independência da aplicação dos dispositivos de entrada e até mesmo de estilos particulares de diálogo. Como o código da aplicação está muito afastado do diálogo de entrada, existe muito mais flexibilidade para adaptar a interface a uma nova configuração de ambiente. [...] A mudança de um dispositivo que utiliza menus do tipo 'pop-up' para outro que não os utiliza demandaria grandes mudanças na especificação de um diálogo se o **UIDS** usasse redes de transição ou gramáticas. No nosso caso, o código da aplicação não iria precisar de nenhuma modificação. [...] Cada adaptação demandaria modificações do sistema de execução, mas isto é muito mais simples do que modificar todas as aplicações ou a especificação das suas interfaces."
- "Outra vantagem do estilo de especificação é a sua capacidade de ser estendido. Numa interface codificada manualmente, não há conhecimento nenhum da estrutura do diálogo. Num sistema baseado em gramáticas ou redes de transição, o único conhecimento disponível é a tradução entre uma sequência de dados de entrada e as chamadas das rotinas semânticas. Como no nosso caso o **UIDS** controla a estrutura concreta dos comandos, podem ser implementados recursos mais completos." (No trecho seguinte, que

omitimos, descreve um sistema de ajuda e um sistema de macros, entre outras possibilidades.)

- "O uso de uma sintaxe específica para os diálogos tem vantagens e desvantagens. Se por um lado limita de maneira bastante drástica os tipos de diálogos que podem ser construídos, pelo outro a criação da interface torna-se muito mais rápida. [...] A natureza restrita das interfaces criadas garante uma uniformidade e consistência de estilo, não somente dentro de uma aplicação, mas entre aplicações diferentes. Esta consistência é obtida gratuitamente ao invés de ser o resultado do esforço consciente de uso de um conjunto significativo de normas para a construção de interfaces."

Podemos concluir que essas vantagens não são específicas de **MIKE** ou **MITUI**, mas comuns a todos os **UIDS** de geração automática.

5.8. Extensões previstas para MITUI

Nós limitaremos a descrever aqui as extensões de **MITUI** que têm ao mesmo tempo um impacto conceitual significativo e viabilidade no mercado, deixando de lado pequenas alterações que não têm impacto no modelo conceitual (como, por exemplo, a possibilidade de as caixas de diálogo serem movimentadas pelo usuário) ou alterações significativas comercialmente inviáveis (como, por exemplo, a criação de uma linguagem de programação própria para evitar o uso da linguagem C).

5.8.1. Orientação a objetos

Como **MITUI** foi projetado usando o paradigma da orientação a objetos, existe um mapeamento natural entre as funções fornecidas por

MITUI e os métodos de uma classe. Por exemplo, todas as funções cujo nome termina em **Create** correspondem a métodos construtores.

Com a popularização da orientação a objetos e a disponibilidade mais ampla de compiladores C++, torna-se possível criar uma versão de **MITUI** orientada a objetos.

```
static void TextNotFound() {  
    dialogBox *THIS = DialogBoxCreate(  
        "Procura Falhou", 9L, 20L, 4L, 38L);  
  
    DialogBoxText(THIS, 0L, 1L,  
        "O texto procurado não foi encontrado", Blue);  
    DialogBoxButton(THIS, 1L, 14L, "&Ok", NULL, TRUE);  
    (void) DialogBoxRun(&THIS);  
}
```

Figura 5.14 - Exemplo de código com a versão atual de **MITUI**

Para tanto não é necessário rescrever o código atualmente existente. É suficiente criar um conjunto de "wrapper classes"* que tornem disponíveis os mesmos recursos usando a notação da orientação a objetos. Compare por exemplo as Figuras 5.14 e 5.15: a primeira exhibe o código necessário com a versão atual para criar uma caixa de diálogo, enquanto a segunda exhibe o código correspondente usando orientação a objetos. Note como a segunda versão é mais clara e mais curta.

---

* Literalmente, *classes que embrulham* em inglês. Não conhecemos um termo adequado ao conceito em português.

```
static void TextNotFound() {  
    DialogBox THIS("Procura Falhou", 9L, 20L, 4L, 38L);  
    THIS.Text(0L, 1L,  
             "O texto procurado não foi encontrado");  
    THIS.Button(1L, 14L, "&Ok");  
    THIS.Run();  
}
```

Figura 5.15 - Exemplo de código usando orientação a objetos

A implementação em C++ da classe necessária para poder escrever o código da Figura 5.14 como aparece na Figura 5.15, corresponde a um esforço de desenvolvimento relativamente modesto, se comparado com o esforço de desenvolvimento da versão inicial. É necessário apenas criar as "wrapper classes" e código de teste para a nova versão (por exemplo, uma versão orientada a objetos do **MITUI Editor**).

5.8.2. Ambientes gráficos

Outra evolução prevista para **MITUI** é a criação de uma versão para o ambiente Windows. Será necessário criar uma nova versão da biblioteca **MITUI**, baseada no modelo de mensagens usado pelo já citado SDK do Windows. Está previsto que este projeto esteja concluído até meados de 1993.

Note que neste passo será preservado não apenas o código fonte das aplicações, mas também o código fonte das "wrapper classes" acima citadas, possibilitando o lançamento simultâneo da versão procedural e da versão orientada a objetos.

A utilização de um ambiente gráfico cria, por sua vez, novas possibilidades de extensão para o **UIDS MITUI**, que poderão ser implementadas a mais longo prazo. Por exemplo, poderão ser criados

recursos para permitir às aplicações lidar com objetos gráficos dentro das janelas de documento.

5.8.3. Gerador interativo de interfaces

Para tornar MITUI mais completo como UIDS, é desejável que exista um programa utilitário que permita ao programador especificar os elementos da interface da aplicação de maneira interativa. Este utilitário deveria emular a execução dos elementos para que o programador os verificasse, e deveria gerar o código necessário para a implementação definitiva. No entanto, o custo de desenvolvimento deste tipo de ferramenta é relativamente elevado.

CAPÍTULO 6

OUTROS ASPECTOS

SOBRE

INTERFACES DE

USUÁRIO

Neste Capítulo teceremos algumas breves considerações sobre alguns aspectos relacionados com interfaces de usuário que não foram tratados nos Capítulos anteriores, mas que, devido à sua importância, não podem ser omitidos.

6.1. Atualização de interfaces já existentes

Quase todas as pesquisas já efetuadas no terreno das interfaces de usuário partem da hipótese que está sendo construída uma interface nova, para um software novo. Os resultados dessas pesquisas fornecem subsídios sobre como produzir novas interfaces de usuário com qualidade.

No entanto, o projetista que está atualizando um software já existente, com uma interface já existente, tem necessidades diferentes.

Teoricamente é possível criar uma interface nova, abandonando a antiga, mas esta alternativa não é viável no mercado de software comercial. É necessário provar aos usuários do software já existente que eles **precisam** de uma nova interface.

Fatores como mudanças tecnológicas, novos equipamentos, novos sistemas operacionais, as limitações da interface antiga e a pressão de produtos concorrentes são fatores que contribuem para a necessidade de atualização de um software já existente, mas nenhum destes fatores, se considerado isoladamente, torna a atualização imprescindível para o usuário.

Ao projetar a nova versão de um software, o projetista precisa manter a consistência com a interface já existente, já que a maioria dos usuários não faz distinção entre a funcionalidade e a interface do software. Entretanto, praticamente não existem resultados de pesquisa nessa área.

Veja [TELLES 90] para uma descrição dos problemas concretos enfrentados na atualização da interface do processador de textos WordStar para o ambiente Windows.

6.2. As interfaces, o ensino e a sociedade

Ainda não existe nenhuma possibilidade de formação para profissionais de computação na área de interfaces de usuário. Mesmo a nível de departamentos de ciência de computação não existem cursos sobre o assunto.

A questão central a ser decidida é (veja [WINOGRAD 90]) determinar se os resultados das pesquisas na área levaram a resultados que possam ser ensinados de forma sistemática.

Winograd questiona a tendência dos departamentos de ciência de computação em isolar a sua pesquisa e ensino quase que exclusivamente nos aspectos técnicos da computação, deixando de lado os aspectos do contexto humano cruciais para projetos de interface bem sucedidos. Ele afirma que "as pessoas e as organizações são vistas como uma parte desafortunadamente desorganizada e imprevisível do ambiente tecnológico".

Winograd apresenta uma proposta de curso experimental, com conteúdos incluindo conhecimentos de filosofia, psicologia, antropologia, ciências sociais e teoria de organização, além dos aspectos tradicionais de ciência de computação. Esse curso seria centrado na vivência prática do processo de desenvolvimento de projetos, ao invés de ser baseado em conteúdos formais. O professor teria o papel de *treinador*, no sentido usado no âmbito esportivo.

Em outra parte do mesmo artigo, Winograd afirma que os cientistas de computação "estão começando a perceber a diferença entre a **eficiência**, no sentido tradicional usado em ciência de computação, e a **produtividade**, no sentido prático de obter mais valor do trabalho".

Em resumo, podemos afirmar que além de a área ainda não ter maturidade suficiente para poder definir conteúdos que possam ser estudados sistematicamente, ela está levando a um questionamento do papel social da ciência da computação.

CAPÍTULO 7

CONCLUSÕES

Este trabalho apresentou uma visão geral dos sistemas para o desenvolvimento de interfaces de usuário, incluindo aspectos formais e práticos, e o sistema **MITUI**, uma ferramenta comercial implementada pelo autor.

MITUI possui várias características únicas e originais que o diferenciam das demais ferramentas. Por exemplo, nenhuma outra ferramenta de interface fornece a mesma quantidade de características automaticamente para a aplicação.

A interface de usuário implementada por **MITUI** é uma interface complexa, com detalhes de acabamento que ultrapassam o nível de qualidade encontrado em muitos softwares comerciais, mesmo de grandes empresas internacionais. As janelas, por exemplo, têm alguns atributos fornecidos automaticamente por **MITUI** (como por exemplo a subdivisão) que nem mesmo o Windows fornece (ele exige que o desenvolvedor da aplicação os implemente).

Mas a principal característica original de **MITUI** faz parte de seu nome: "The first high-level CUA API". Os recursos disponibilizados por **MITUI** para as aplicações têm um nível semântico muito alto, o que leva a uma API composta por menos de 80 funções, enquanto nenhum dos produtos concorrentes existentes no mercado internacional utiliza menos que algumas centenas de funções. Esse fato rendeu a **MITUI** uma cobertura jornalística na imprensa internacional especializada em informática não atingida antes por nenhum outro software brasileiro. Entre outros, merecem destaque a

reportagem de outubro de 1991 da revista "Program Now" [PROG NOW 91], principal revista de desenvolvimento de software da Grã Bretanha, e a notícia publicada na seção internacional da revista Byte, editada nos Estados Unidos, em maio de 1992 [BYTE 92].

Finalmente, o fato de a primeira implementação de MITUI ter como alvo PCs rodando sob MS-DOS no modo texto, permite o acesso de usuários e desenvolvedores brasileiros à tecnologia de interfaces de usuário com os equipamentos mais freqüentemente disponíveis: Compatíveis do tipo XT.

BIBLIOGRAFIA

1. [AHO 72]
Aho, Alfred V. & Ullman, Jeffrey D.
The Theory of Parsing, Translation, and Compiling
Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1972
2. [APPLE 85]
Apple Computer
Inside Macintosh, Vol. 1
Addison-Wesley Publishing Company, Reading, Massachusetts, USA,
1985
3. [ASU 86]
Aho, Alfred V.; Sethi, Ravi & Ullman, Jeffrey D.
Compilers, Principles, Techniques, and Tools
Addison-Wesley Publishing Company, Reading, Massachusetts, USA,
1986
4. [BARTH 86]
Barth, Paul S.
An Object-Oriented Approach to Graphical Interfaces
ACM Transactions on Graphics, 5(2), Abril 1986, 142-172
5. [BOBROW 86]
Bobrow, D.G.; Mittal, S. & Stefik, M.J.
Expert Systems: Perils and Promise
Communications of the ACM, 29(9), Setembro 1986, 880-894

6. [BUXTON 83]
Buxton, William; Lamb, M.R.; Sherman D. & Smith, K.C.
Towards a comprehensive user interface management system
Computer Graphics, 17(3), Julho 1983, 35-42
7. [BUXTON 86]
Buxton, William & Myers, Brad A.
A Study in Two-Handed Input
Proceedings of SIGCHI'86, ACM, New York, USA, 1986, 321-326
8. [BYTE 92]
What's New - International
High-Level CUA API
Byte, 17(5), Maio 1992, 84IS-24
9. [CARDELLI 85]
Cardelli, Luca & Pike, Rob
Squeak: a Language for Communicating with Mice
Computer Graphics, 19(3), Julho 1985, 199-204
10. [CARDELLI 87]
Cardelli, Luca
Building User Interfaces by Direct Manipulation
Research Report 22, Digital Equipment Corp., Systems Research
Center, Palo Alto, California, USA, 1987
11. [CHIA 85]
Chia, M.S.
*An event based dialogue specification for automatic generation of
user interfaces*
M.Sc. thesis, Dept. of Computing Science, University of Alberta,
Edmonton, Alberta, Canadá, 1985

12. [CONKLIN 87]
Conklin, Jeff
Hypertext: An Introduction and Survey
Computer, Setembro 1987, 17-41
13. [CORMACK 88]
McCormack, John & Asente, P.
An Overview of the X Toolkit
Procedures of the ACM SIGGraph Symposium on User-Interface
Software, ACM, New York, USA, 1988, 46-55
14. [EDMONDS 77]
Edmonds, E.A. & Guest, S.P.
An interactive tutorial system for teaching programming
Proceedings of the IEEE Conference 36 - Computer Systems and
Technology (Brighton, Inglaterra, 1977), Institute of Electrical and
Radio Engineers, Londres, 263-270
15. [ELSHOFF 74]
Elshoff, E.L.; Beckermeyer, R.; Dill, J.; Marcotty, M. & Murray, J.
Handling asynchronous interrupts in a PL/I-like language
Software Practice and Experience, 4, 1974, 117-124
16. [FISHER 87]
Fisher, G.L. & Joy, K.I.
*A Control-Panel Interface for Graphics and Image-Processing
Applications*
SIGCHI '87 Procedures, ACM, New York, 1987, 285-290
17. [FLECHIA 87]
Flechchia, M.A. & Bergeron, R.D.
Specifying Complex Dialogs in ALGEA
Procedures of SIGCHI+GI'87, ACM, New York, USA, 1987, 229-234

18. [FOLEY 82]
Foley, James D. & Van Dam, Andries
Fundamentals of Interactive Computer Graphics
Addison-Wesley Publishing Company, Reading, Massachusetts, 1982,
217-243
19. [FOLEY 87]
Foley, James D.
*Transformations on a Formal Specification of User-Computer
Interfaces*
Computer Graphics, 21(2), Abril 1987, 109-113
20. [GAITH 82]
Proceedings of
Human Factors in Computing Systems
ACM, Gaithersburg, Maryland, USA, 1982
21. [GOLDBERG 83]
Goldberg, Adele & Robson, D.
Smalltalk-80: The Language and Its Implementation
Addison-Wesley Publishing Company, Reading, Massachusetts, 1983
22. [GOOD 84]
Good, Michael D.; Whiteside, John A.; Wixon, Dennis R. & Jones,
Sandra J.
Building a User-Derived Interface
Communications of the ACM, 27(10), Outubro 1984, 1032-1043
23. [GREEN 79]
Green, Mark
A graphical input programming system
M.Sc. thesis, Dept. of Computer Science, University of Toronto,
Toronto, Canada, 1979

24. [GREEN 85]
Green, Mark
The University of Alberta user interface management system
SIGGraph '85 Proceedings, Computer Graphics, 19(3), Julho 1985, 205-213
25. [GREEN 86]
Green, Mark
A Survey of Three Dialogue Models
ACM Transactions on Graphics, 5(3), Julho 1986, 244-275
26. [GRUDIN 90]
Grudin, Jonathan
The computer reaches out: the historical continuity of interface design
CHI'90 Proceedings (número especial do SIGCHI Bulletin), Abril 1990, 261-268
27. [GUEST 82]
Guest, S.P.
The use of software tools for dialogue design
International Journal of Man-Machine Studies, 16, 1982, 263-285
28. [HANAU 80]
Hanau, P.R. & Lenorovitz, D.R.
Prototyping and simulation tools for user/computer dialogue design
SIGGraph '80 Proceedings, Computer Graphics, 14(3), Julho 1980, 271-278
29. [HAREL 87]
Harel, D.
STATECHARTS: A Visual Formalism for Complex Systems
Science of Computer Programming, 8(3), Junho 1987, 231-274

30. [HAYES 85]
Hayes, P.J.; Szekely, P.A. & Lerner, R.A.
Design Alternatives for User-Interface Management Systems Based on Experience with Cousin
Procedures do SIGCHI '85, ACM, New York, 1985, 169-175
31. [HENDERS 86]
Henderson Jr, D.A.
The Trilium User-Interface Design Environment
SIGCHI '86 Procedures, ACM, New York, 1986, 221-227
32. [HILL 86]
Hill, Ralph D.
Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction — The Sassafras UIMS
ACM Transactions on Graphics, 5(3), Julho 1986, 179-210
33. [IBM 83]
International Business Machines Corp.
Intellect Reference Manual
Program offering LY20-9083-0
USA, 1983
34. [IBM 89]
International Business Machines Corp.
Systems Application Architecture: Common User Access, Advanced Interface Design Guide
IBM Boca Raton, Florida, USA, 1989
35. [JACOB 86]
Jacob, Robert J.K.
A Specification Language for Direct-Manipulation User Interfaces
ACM Transactions on Graphics, 5(4), Outubro 1986, 283-317

36. [JOHNSON 77]
JOHNSON, S.C.
Yacc - yet another compiler compiler
Computing Science Technical Report 32, AT&T Bell Laboratories,
Murray Hill, New Jersey, 1977
37. [KASIK 82]
Kasik, David J.
A User Interface Management System
Computer Graphics, 6(3), Julho 1982, 99-106
38. [LOMET 73]
Lomet, D. B.
A formalization of transition network systems
Journal of the ACM, 20(2), Abril 1973, 235-257
39. [MBI 91]
Mayer & Bunge Informática S/C Ltda.
"MITUI - The first high-level CUA API"'s Tutorial Manual
São Paulo, São Paulo, Brasil, 1991
40. [MICROSOFT 90]
Microsoft Corporation
Microsoft Windows Guide to Programming
Microsoft Press, Redmond, Washington, USA, 1990
41. [MICROSOFT 92]
Microsoft Corporation
Windows Application Style Guide
Microsoft Press, Redmond, Washington, USA, 1992

42. [MYERS 86]
Myers, Brad A. & Buxton, William
Creating Highly-Interactive and Graphical User Interfaces by Demonstration
Computer Graphics, 20(4), Agosto 1986, 249-258
43. [MYERS 89]
Myers, Brad A.
User-Interface Tools: Introduction and Survey
IEEE Software, Janeiro 1989, págs.15-23
44. [MYERS 91]
Myers, Brad A. & Rosson, Mary B.
User Interface Programming Survey
SIGCHI Bulletin, 23(2), Abril 1991, 27-30
45. [NAKA 91]
Nakatsuyama, Hisashi; Murata, Makoto & Kusumoto, Koji
A new framework for separating user interfaces from application programs
SIGCHI Bulletin, 23(1), Janeiro 1991, 88-91
46. [NEWMAN 68]
Newman, W.M.
A system for interactive graphical programming
Proceedings of the Spring Joint Computer Conference, Atlantic City, New Jersey, USA, Abril 1968, 47-54
47. [OLSEN 83]
Olsen, Dan R. Jr.
SYNGRAPH: A Graphical User Interface Generator
Computer Graphics, 17(3), Julho 1983, 43-50

48. [OLSEN 84]
Olsen, Dan R. Jr.
Pushdown automata for user interface management
ACM Transactions on Graphics, 3(3), Março 1984, 177-203
49. [OLSEN 85]
Olsen, Dan R. Jr; Dempsey, Elizabeth P. & Rogge, Roy
Input/Output Linkage in a User Interface Management System
Computer Graphics, 17(3), Julho 1985, 191-197
50. [OLSEN 86]
Olsen, Dan R. Jr
MIKE: The Menu Interaction Kontrol Environment
ACM Transactions on Graphics, 5(4), Outubro 1986, 318-344
51. [OLSEN 87]
Olsen, Dan R. Jr
Larger Issues in User-Interface Management
Computer Graphics, 19(2), Abril 1987, 134-137
52. [PARNAS 69]
Parnas, David L.
On the use of transition diagrams in the design of a user interface for an interactive computer system
Proceedings of the 24th National ACM Conference, San Francisco, California, USA, 1969, 379-385
53. [PFAFF 85]
Pfaff, Günther E., Editor
User Interface Management Systems
Springer-Verlag, Berlin, 1985

54. [PIMENTA 91]
Pimenta, Marcelo S. & Heuser, Carlos A.
Canonicus: Um modelo para portabilidade de Programas Interativos
Anais do V Simpósio Brasileiro de Engenharia de Software, Ouro
Preto (MG), Outubro 1991, 119-133
55. [PLAUGER 90]
Plauger, J. P. & Brodie, Jim
Standard C
Microsoft Press, Seattle, Washington, USA, 1990
56. [PROG NOW 91]
Out of the Box
Make mine MITUI
ProgramNOW, 5(10), Outubro 1991, 42-43
57. [ROSSON 87]
Rosson, M.B.; Maass, S. & Kellogg, W.A.
*Designing for Designers: An Analysis of Design Practices in the Real
World*
SIGCHI '87 Procedures, ACM, New York, 1987, 137-142
58. [SCHMUCK 86]
Schmucker, K.J.
MacApp: an Application Framework
Byte, 11(8), Agosto 1986, 189-193
59. [SCHULER 85]
Schulert, A.J.; Rogers, G.T. & Hamilton, J.A.
ADM: A Dialogue Manager
Procedures do SIGCHI '85, ACM, New York, 1985, 177-183

60. [SETZER 83]
Setzer, Valdemar W. e Homem de Melo, Inês
A construção de um compilador
Editora Campus, Rio de Janeiro, 1986, 3ª edição
61. [SETZER 89]
Setzer, Valdemar W. & Mayer, Roberto C.
Grafos sintáticos simples e um grafo para a linguagem C ANSI
Relatório Técnico MAC-8905, Instituto de Matemática e Estatística, USP, 1989
62. [SIBERT 86]
Sibert, John L.; Hurley, William D. & Bleser, Teresa W.
An Object-Oriented User Interface Management System
Computer Graphics, 20(4), Agosto 1986, 259-268
63. [SWART 82]
Swartout, W. & Balzer, R.
The Inevitable Intertwining of Specification and Implementation
Communications of the ACM, 25(7), Julho 1982, 438-440
64. [SZEKELY 88]
Szekely, Peter & Myers, Brad A.
A User-Interface Toolkit Based on Graphical Objects and Constraints
SIGPlan Notices, Novembro 1988, 36-45
65. [TELLES 90]
Telles, Marcy
Updating an Older Interface
CHI'90 Proceedings (número especial do SIGCHI Bulletin), Abril 1990, 243-247

66. [WASSERMANN 82]
Wassermann, A.I. & Shewmake, D.T.
Rapid Prototyping of Interactive Information Systems
SIGSoft Software Engineering Notes, Dezembro 1982, 171-180
67. [WINOGRAD 90]
Winograd, Terry
What can we teach about human-computer interaction
CHI'90 Proceedings (número especial do SIGCHI Bulletin), Abril 1990, 443-449
68. [WIRTH 85]
Wirth, Niklaus
From Programming Language Design To Computer Construction
Turing Award Lecture, Communications of the ACM, 28(2), Fevereiro 1985, 160-164
69. [WOODS 70]
Woods, W.A.
Transition networks for natural language analysis
Communications of the ACM, 13(10), Outubro 1970, 591-606