


# Construção de Interfaces Homem-Computador

*O uso de Estadogramas na especificação e implementação  
de controle de interface*

Este exemplar corresponde a redação final da  
tese devidamente corrigida e defendida pelo Sr.  
*Fábio Nogueira de Lucena* e aprovada pela Co-  
missão Julgadora.

Campinas, 25 de março de 1993.

  
PROF. DR. HANS KURT E. LIESENBERG  
orientador  
*Hans K. E. Liesenberg*

Dissertação apresentada ao Instituto de Matemática,  
Estatística e Ciência da Computação, UNICAMP,  
como requisito parcial para a obtenção do Título de  
Mestre em Ciência da Computação.

# Construção de Interfaces Homem-Computador

## *O uso de Estadogramas na especificação e implementação de controle de interface*

*“A interface é freqüentemente o mais importante fator para determinar o sucesso ou falha de um sistema, além de ser um dos mais caros.”*  
Baecker e Buxton[BB87, pág. 1]

### Resumo

Existem várias técnicas para especificação e implementação do controle de interfaces homem-computador, i.e., técnicas para descrição e implementação da sintaxe permitida das ações do usuário, das reações do computador e como o diálogo (entre homem e computador) evolui ao longo do tempo. As técnicas, contudo, ainda apresentam inconvenientes. Este trabalho concentra-se na representação e implementação desta sintaxe.

Estadogramas (*statechart*, neologismo já usado em outros trabalhos) apresentam indícios de serem adequados para descreverem este comportamento. São diagramas que estendem os diagramas de transição de estados convencionais e eliminam inconvenientes dos últimos.

O uso dos Estadogramas no desenvolvimento de uma interface real permitiu identificar mudanças que tornam estes diagramas mais apropriados para este emprego específico. O uso mostrou que Estadogramas precisam de recursos para tratamento da apresentação de uma interface e de outras adaptações sugeridas. A observação do código gerado por uma ferramenta, que implementa Estadogramas, ainda permitiu identificar elementos desejáveis quanto a estrutura do código a ser produzida.

### Abstract

A variety of techniques exists for the specification and the implementation of human-computer interface control, i.e., techniques to describe and implement the syntax of user's actions, of the reactions of a computer and of how the dialogue between a user and a computer evolves along a period of time. These techniques, however, present some drawbacks. This work concentrates on representation and implementation of a dialogue syntax based on the statechart notation.

A statechart seems suitable to describe this kind of behaviour. It extends state transition diagrams and overcomes some of the shortcomings of the later.

The use of the statechart notation in the development of a realistic interface led to improvements which have been made in order to apply it specifically in this context. This use showed the need of some kind of supported at the presentation level and of some changes of notation. The observation of the code generated by an already existing tool to implement statechart behaviour give us some insights as well about desired elements in relation to the structure of the generated code.

# Agradecimentos

Ao Criador, pela permissão.

Ao orientador Dr. Hans Kurt E. Liesenberg pela disponibilidade e atenção infindáveis.

Ao CNPq (Projetos ETHOS e RNP).

Ao Rodolfo Miguel Baccarelli, ouvinte incansável e apoio inestimável.

Ao Thierson, Tutumi, Fábio Costa, Mário, Marcus, que apoiaram direta ou indiretamente.

Agradeço à Profa. Ana Lúcia C. Cavalcanti (UFPE) pela prontidão em solucionar dúvidas.

Aos colegas de trabalho (UFG) que reconheceram a necessidade desta realização.

# Dedicação

À toda a minha família.

Em especial meus pais. Sem eles este trabalho não seria possível: *Nogueira e Tereza.*

Minhas irmãs: *Núbia, Nútia e Flávia.*

À minha namorada, *Lílian.*

A todos meus sinceros agradecimentos pela compreensão e apoio.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Conceito e importância de interfaces . . . . .	1
1.2	O problema . . . . .	5
1.3	Este trabalho . . . . .	7
1.4	Objetivos . . . . .	10
1.5	Motivação . . . . .	11
1.6	Visão geral da dissertação . . . . .	13
1.7	Convenções . . . . .	14
1.8	Resumo . . . . .	14
<b>2</b>	<b>Estadogramas</b>	<b>15</b>
2.1	Origem . . . . .	15
2.2	Qual o problema com sistemas reativos? . . . . .	18
2.3	Uma taxonomia para sistemas reativos . . . . .	19
2.4	Relação entre sistemas reativos e modelos de interface . . . . .	19
2.5	O que são Estadogramas? . . . . .	20
2.6	Descrição informal . . . . .	21
2.7	Poder de expressão . . . . .	28
2.7.1	Estadogramas, uma extensão dos diagramas de estados . . . . .	28
2.8	Implementação . . . . .	30
2.8.1	Hipótese síncrona . . . . .	30
2.8.2	Comportamento determinístico . . . . .	31
2.9	Ferramentas . . . . .	32
2.9.1	Statemaster . . . . .	33
2.9.2	Statemate . . . . .	33
2.9.3	Reacto . . . . .	33
2.9.4	Tradutor Blob e Egrest . . . . .	34
2.10	Resumo . . . . .	39
<b>3</b>	<b>Interface Homem-Computador</b>	<b>41</b>
3.1	Terminologia básica . . . . .	41
3.2	Estilos de Interação . . . . .	44

3.3	Modelos . . . . .	46
3.4	Uma taxonomia para o software . . . . .	49
3.5	Engenharia de software para interfaces . . . . .	50
3.5.1	Qualidades em uma interface . . . . .	51
3.5.2	Princípios para garantir a qualidade . . . . .	51
3.5.3	Técnicas . . . . .	52
3.5.4	Metodologias . . . . .	55
3.5.5	Portabilidade . . . . .	55
3.5.6	Protótipos . . . . .	55
3.5.7	Ciclo de vida . . . . .	56
3.6	Ferramentas . . . . .	61
3.7	Requisitos para o controlador do diálogo . . . . .	64
3.8	Interfaces e o Paradigma de Objetos . . . . .	65
3.9	Resumo . . . . .	66
<b>4</b>	<b>Desenvolvendo uma Interface</b> . . . . .	<b>67</b>
4.1	Um Sistema . . . . .	67
4.2	Visão geral do desenvolvimento do sistema . . . . .	70
4.3	Uma Interface . . . . .	70
4.4	Visão geral do desenvolvimento da interface . . . . .	76
4.5	Projeto . . . . .	76
4.6	Implementação . . . . .	78
4.7	Ambiente de Desenvolvimento . . . . .	79
4.8	Resumo . . . . .	81
<b>5</b>	<b>Propondo alterações</b> . . . . .	<b>83</b>
5.1	Especificação formal . . . . .	83
5.2	Algumas propostas . . . . .	84
5.2.1	Modo . . . . .	84
5.2.2	<i>Callbacks functions</i> . . . . .	87
5.2.3	Variáveis . . . . .	89
5.2.4	Depuração . . . . .	90
5.2.5	Reutilização de comportamentos . . . . .	90
5.2.6	Alteração dinâmica do comportamento . . . . .	91
5.2.7	<i>Undo</i> . . . . .	91
5.2.8	Escopo de eventos . . . . .	92
5.2.9	Estabelecendo prioridades . . . . .	94
5.3	Uma notação em forma de texto para Estadogramas . . . . .	95
5.3.1	Transições . . . . .	95
5.3.2	Transição alternativa . . . . .	97
5.4	Implementação e suas implicações . . . . .	98
5.4.1	Manipulação direta . . . . .	99

5.4.2	Díálogo <i>multithread</i> . . . . .	100
5.5	Resumo . . . . .	101
<b>6</b>	<b>Conclusão</b>	<b>103</b>
6.1	Contribuições da tese . . . . .	105
6.2	Trabalho futuro . . . . .	107
6.3	Considerações finais . . . . .	107
<b>A</b>	<b><i>Windows</i></b>	<b>117</b>
A.1	Caracterizando o ambiente <i>Windows</i> . . . . .	117
A.2	Visão geral do funcionamento . . . . .	118
A.3	Paradigma de objetos e <i>Windows</i> . . . . .	118
A.4	Construindo aplicações <i>Windows</i> . . . . .	119
A.5	Referências para o programador . . . . .	120

# Lista de Figuras

1.1	Visão de interface da perspectiva de um observador externo[Chi85]. . . . .	2
1.2	Área de interesse deste trabalho. . . . .	9
2.1	Sistema transformacional. . . . .	16
2.2	Sistema reativo. . . . .	17
2.3	Relacionando modelos. . . . .	20
2.4	Tipos de estados em Estadograma. . . . .	22
2.5	Estados concorrentes. . . . .	23
2.6	Estabelecendo prioridades. . . . .	24
2.7	Arestas em Estadograma. . . . .	25
2.8	Transição entre estados. . . . .	26
2.9	Aresta que não causa transição. . . . .	26
2.10	Exemplo de <i>History</i> simples. . . . .	27
2.11	Exemplo de <i>History H*</i> . . . . .	27
2.12	Estadogramas X diagramas de estados convencionais . . . . .	29
2.13	Exemplificando o determinismo. . . . .	31
2.14	Outro exemplo de determinismo. . . . .	32
2.15	Ciclo simples de utilização do tradutor. . . . .	34
2.16	Relação entre Tradutor e EgreSt. . . . .	35
2.17	Arquitetura de código. . . . .	36
2.18	Nova arquitetura de código. . . . .	37
3.1	Visão simplificada de um sistema interativo. . . . .	42
3.2	Modelo de Protocolo Virtual. . . . .	47
3.3	Modelo de Seeheim. . . . .	49
3.4	Arquitetura de software para interfaces. . . . .	49
3.5	Independência de diálogo . . . . .	53
3.6	Independências de diálogo e dados em um sistema[WPSK86]. . . . .	53
3.7	Modelo em Cascata do ciclo de vida de <i>software</i> . . . . .	57
3.8	Ciclo de vida estrela para o desenvolvimento de interfaces. . . . .	58
3.9	Participantes de uma interação homem-computador. . . . .	64
4.1	Arquitetura do Micromundo Musical. . . . .	68



4.2	Árvore . . . . .	69
4.3	Um simples estado. . . . .	71
4.4	A primeira tela . . . . .	72
4.5	Estado <b>Operação</b> . . . . .	73
4.6	Estilos de diálogo disponíveis. . . . .	74
4.7	Modelando a operação de <i>zoom</i> . . . . .	74
4.8	Um instante de interação com o sistema interativo . . . . .	75
4.9	Uma simples especificação. . . . .	75
4.10	Ferramentas utilizadas. . . . .	79
4.11	Fornecendo mais recursos à notação de Estadogramas. . . . .	82
5.1	Objeto de interação[Jac86]. . . . .	85
5.2	Usando estado para modelar processo. . . . .	86
5.3	Usando estado para modelar o que o usuário percebe. . . . .	87
5.4	Comunicação entre componentes de sistemas interativos. Adaptado de [Sin89].	89
5.5	Uma proposta para <i>undo</i> ( $undo(tr) = tr2 + ações$ ). . . . .	92
5.6	Definindo contexto para eventos. . . . .	93
5.7	Escopo de eventos. . . . .	94
5.8	Concorrência em níveis distintos. . . . .	94
5.9	Transição envolvendo conjuntos de estados. . . . .	97
5.10	Transições: <i>pseudo</i> , <i>default</i> e $\lambda$ . . . . .	99
A.1	Triagem de mensagens no <i>Windows</i> . . . . .	119
A.2	Construindo uma aplicação <i>Windows</i> . . . . .	120

# Lista de Tabelas

3.1	Operações básicas do módulo EDITOR. . . . .	56
3.2	Vantagens e desvantagens de metodologias para projeto de interface. . . . .	62

\* \* \* \* \*

Ada<sup>TM</sup> é marca registrada do governo norte-americano (USA).  
MacApp<sup>TM</sup> e Macintosh<sup>TM</sup> são marcas registradas da *Apple Computer, Inc.*.  
MS-DOS<sup>TM</sup> e Microsoft *Windows*<sup>TM</sup> são marcas registradas da *Microsoft Corporation*.  
Smalltalk-80<sup>TM</sup> é marca registrada da *Xerox Corporation*.  
UNIX<sup>TM</sup> é marca registrada da AT&T.

# Capítulo 1

## Introdução

Acerca da frustração parcial do Word v2.0, Bill Gates, grande acionista da Microsoft, disse:  
*“Merecemos a culpa por não termos facilitado o seu aprendizado. No tocante aos recursos, o produto era fantástico, mas no que se refere a facilidade dos primeiros passos, não nos saímos muito bem.”*  
Ichbiah[IK92]

Este capítulo define o termo interface e mostra sua relevância no desenvolvimento de sistemas interativos; destaca ambientes gráficos precursores das características comumente encontradas nas atuais interfaces; identifica problemas com o seu desenvolvimento e localiza o de interesse deste trabalho; apresenta o objetivo e motivações para a consecução deste trabalho e fornece uma visão macroscópica da dissertação. No fim do capítulo encontram-se siglas e convenções usadas.

---

### 1.1 Conceito e importância de interfaces

O que é uma interface? Embora seja um termo comum na computação, não é completamente indevido descrever sua aceção, principalmente pelas várias conotações possíveis.

---

**Interface:** compreende todos os comportamentos do usuário<sup>1</sup> e do computador que são observáveis externamente[Chi85]. Há uma linguagem de entrada, uma de saída para refletir os resultados e um protocolo de interação (veja a figura 1.1).

---

---

<sup>1</sup>O termo usuário refere-se ao usuário final (*end user*) de um sistema, i.e., pessoa para a qual uma interface é desenvolvida. Há quem prefira “operador” a usuário final[BC91, pág. 3].

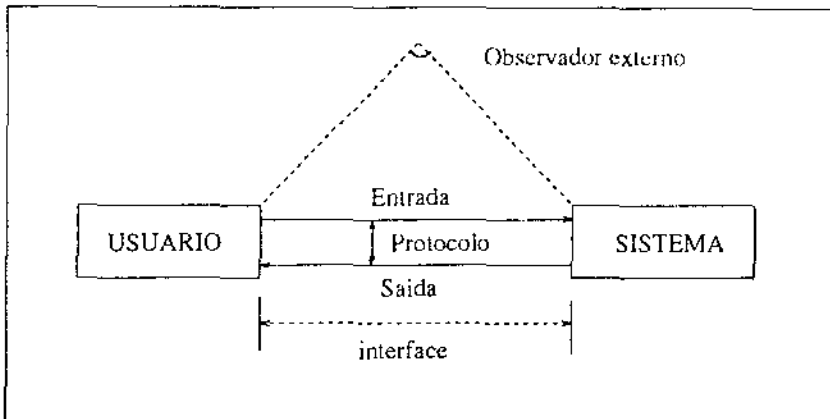


Figura 1.1: Visão de interface da perspectiva de um observador externo[Chi85].

O conceito de interface visto anteriormente é muito abrangente para os propósitos deste trabalho, que se concentra em engenharia de software. Refinando a definição anterior, i.e., sendo mais preciso:

---

**Interface:** software de um sistema interativo responsável por traduzir ações do usuário em ativações das funcionalidades do sistema, permitir que os resultados possam ser observados e coordenar esta interação.

---

O capítulo 3 fornece mais detalhes da separação entre a interface e a aplicação. Neste trabalho aplicação refere-se a funcionalidade de um sistema interativo. A interface é o componente do sistema interativo responsável por permitir que o usuário tenha acesso a esta funcionalidade. Recomenda-se [Har89] para comentários mais exaustivos acerca desta separação.

Usa-se por todo o texto o termo diálogo como sinônimo de interface. Em muitos casos, diálogo é usado para designar a interação entre o usuário e o computador, ou seja, um conjunto de “ações” de ambas as partes. Tais ações nem sempre são visíveis, por exemplo, validar entradas fornecidas, calcular posições dos nós de uma árvore para que possam ser impressas e assim por diante. A segunda e mais precisa definição de interface inclui estes processamentos.

Newman e Sproull[NS79] há mais de uma década já descreviam a relevância das interfaces no sucesso de um software e os problemas relacionados com sua construção. Aprender a usá-las geralmente implica em investimento de tempo razoável. Uma boa interface torna o programa mais fácil de aprender e usar,<sup>2</sup> i.e., amigável (*user-friendly*). Em outras palavras,

<sup>2</sup>Do inglês *easy-to-learn* e *easy-to-use*. Geralmente são substituídos por *user-friendly* para caracterizar uma “boa” interface. Habermann[Hab91] sugere uma semântica para *easy-to-learn* e *easy-to-use*: (1) a interface é

a interface pode influir na produtividade do usuário. Construí-las requer o emprego de técnicas e métodos específicos.

Interfaces têm recebido crescente consideração por parte da literatura dirigida para a área de desenvolvimentos de sistemas [Chi85, Hii89a, Hix90]. A conferência de maior destaque acerca de fatores humanos em sistemas de computação<sup>3</sup> obteve, na última edição, um número recorde de artigos submetidos e de participantes, em relação aos anteriores.

Esta crescente atenção não se restringe ao ambiente acadêmico. Curtis em [Cur91] destaca vários projetos japoneses de longo prazo envolvendo consideráveis quantias em dinheiro. Todos com a atenção centralizada no desenvolvimento de interfaces. Segundo Curtis, uma vez atingida certa uniformidade na confiabilidade dos produtos, as interfaces são o próximo passo para a distinção e vantagem comercial, o que justifica os altos investimentos japoneses. Bass e Coutaz [BC91] citam outros fatores que impulsionam esta área do conhecimento: (1) expectativa dos usuários; (2) custos envolvidos no desenvolvimento de interface; e (3) surgimento de novas tecnologias de interação. Conforme [KA90], "gigantes" da computação como a IBM, Microsoft, Digital e Hewlett-Packard (HP) também têm dado grande ênfase para as interfaces.

Em [BB87] é evidenciada a importância e o custo elevado das interfaces, que chegam a ser grande parte do código de um sistema interativo. Em [BC91, pág. v] afirma-se que a interface consome até 70% dos custos totais do ciclo de vida de um sistema interativo, e que este percentual cresce proporcionalmente com a sofisticação da interface. Bobrow [BMS86] relata que para um sistema especialista poder-se-ia esperar que a maior parte de código fosse devotada à base de conhecimento e à máquina de inferência. Entretanto, ambos, a base de conhecimento e a máquina de inferência equivalem a apenas um terço da memória total. A interface é a maior parte do sistema, cerca de 42% do código. Bobrow diz que isto não é uma situação atípica e que outros sistemas apresentam proporções similares. Em recente pesquisa sobre dezenas de projetos Myers [MR92] conclui que 48% do código é dedicado à interface. Ainda cita que, do tempo total de desenvolvimento, aquele consumido na interface equivale a 48% do tempo de projeto de todo o sistema interativo, 50% de toda a implementação e 37% da manutenção.

O sucesso e o custo de um sistema interativo não dependem exclusivamente da complexidade das funções que realiza, mas também da interface com a qual o usuário tem acesso a esta funcionalidade. A qualidade de uma interface tem grande influência no sucesso comercial de um software [SV91, DL91]. Neumann [Neu89] chega a afirmar que "mesmo os melhores sistemas de hardware e software tornam-se ineficazes devido a uma interface imprópria com o usuário."

A seção abaixo fornece um breve histórico da evolução de ambientes gráficos, que influenciaram significativamente as atuais interfaces. Tais ambientes fornecem uma visão do custo

---

invisível (o usuário pode concentrar-se nas tarefas que necessita realizar); (2) é previsível; (3) é flexível e (4) as pessoas gostam delas.

<sup>3</sup> *Human Factors in Computing Systems — CHI'92 Conference Proceedings, Monterey, California.*

de desenvolvimento de interfaces e do sucesso que podem significar.

### Ambientes gráficos

Um marco nítido na história das atuais interfaces está na criação do ambiente *Smalltalk*. Este ambiente permite o uso de várias janelas<sup>4</sup> sobrepostas.<sup>5</sup> As janelas podem ser selecionadas e transladadas na tela com o uso do mouse. A primeira versão deste ambiente, conforme [IK92], foi testada no Alto, um protótipo de computador desenvolvido no Xerox PARC (*Palo Alto Research Center*), um dos computadores mais fáceis de se usar até aquele momento. A interface para o Star, ainda mais sofisticada, foi criada em 1981 e introduziu os ícones (§3.2). Usando conceitos apresentados nestas máquinas e com preço mais acessível surge o *Apple Macintosh* em 1984. Em 1985 é lançado o *Windows* (apêndice A) -- uma tentativa de fornecer aos usuários do computador mais difundido, o IBM-PC, facilidades semelhantes àquelas que os usuários dos computadores supracitados possuíam. *Windows* consumiu 110 mil horas de programação. O resultado foi um sucesso: em dezembro de 1989 já haviam sido vendidas 2 milhões de cópias. No fim de 1990 e início de 1991 eram vendidas 30 mil cópias por semana [IK92]. Grande parte deste sucesso é atribuído, sobretudo, à interface com o usuário. Para contrastar com o tempo de desenvolvimento do *Windows*, a interface para o Star consumiu 6 anos [Cou85]. *Windows NT* é uma promessa de evolução do *Windows*. Embora não tenha sido lançado, até o momento em que este trabalho está sendo escrito, muitas especulações [YS92] já são feitas sobre este produto.

---

<sup>4</sup>Janela (*window*) é uma área geralmente retangular da tela. Através desta área um programa recebe entradas e emite os resultados do seu processamento. Geralmente os ambientes que permitem o uso de janelas são multiprogramados (refere-se à *multiprogramming*). Neste caso cada janela pode estar ligada a um programa executado concorrentemente.

<sup>5</sup>Janelas que podem ter uma interseção não nula, ao invés de janelas "azulejadas" (*tiled*).

## 1.2 O problema

Viu-se que interfaces são partes significativas de sistemas interativos e que absorvem parte notável de todo o tempo de desenvolvimento. Viu-se também que o sucesso de um software interativo depende de sua interface. Esta seção concentra-se nas dificuldades de construção de interfaces, identifica neste conjunto o problema de interesse deste trabalho e por fim fornece detalhes específicos do mesmo.

Baecker e Buxton[BB87, pág. 1] caracterizam muito bem o projeto<sup>6</sup> de interfaces:

“O sucesso ou falha de uma interface é determinado por um complexo elenco de questões relacionadas . . . , inclui se o sistema é agradável ou hostil, fácil ou difícil de aprender, fácil ou difícil de usar, compreensível ou intolerante diante dos erros humanos. As capacidades e disciplinas necessárias para obter um equilíbrio adequado entre esses fatores são tão diversos quanto eles próprios. Entre outros, eles incluem as habilidades do projetista gráfico e industrial, um entendimento de dinâmica de organização e processos, um entendimento de cognição humana, percepção e habilidades, um conhecimento de tecnologia de tela, dispositivos de entrada, técnicas de interação e metodologias de projeto, além de um talento para elegância no projeto do sistema. Projeto de uma interface real é, dessa forma, um processo multidisciplinar que requer um amplo ponto de vista de qualquer problema de projeto. É também uma tarefa que requer mais habilidades do que um único indivíduo geralmente possui.”

Este caráter multidisciplinar do projeto de interfaces tem implicação direta: a sua descrição é uma atividade para especialistas em fatores humanos. O perfil deste projetista é essencialmente orientado para o aspecto psicológico e não para o computacional. Uma inferência imediata pode ser obtida desta situação: ferramentas de apoio ao projeto de interfaces devem ser fáceis de usar, pois nem sempre um programador é a pessoa mais adequada para desenvolver um projeto desta natureza. Muitos trabalhos na literatura ressaltam essa facilidade que ainda falta ser incorporada às atuais ferramentas empregadas [HH89a, HH89b, MR92, TB85, HLS90, DL91, OGL<sup>+</sup>87]. Molich em [MN90] mostra que especialistas em computação acham difícil a identificação de problemas no projeto de uma interface. Em suma, do ponto de vista de projeto tem-se várias áreas do conhecimento envolvidas e, como conseqüência, a necessidade de ferramentas apropriadas para os especialistas destas áreas. As ferramentas existentes, contudo, ainda não atendem às expectativas. São, geralmente, para uso de especialista em computação.

É bom ressaltar que não existem regras que garantam resultados satisfatórios no projeto de interfaces. Isto conduz a outra variável de projeto: a literatura é unânime em considerar o projeto um processo iterativo e centralizado no usuário. Ou seja, há uma necessidade de

---

<sup>6</sup>Entenda como a fase de projeto do ciclo de vida de uma interface (§3.5.7).

construção de protótipos (o processo é iterativo) e constantes testes com o usuário (centralizado no usuário) que fornecem informações para os projetistas refazerem o projeto. O ciclo repete-se até a obtenção de resultados satisfatórios. Good em [GWWJ84] fornece um exemplo com resultados animadores do desenvolvimento centralizado no usuário e em [MR92] vê-se que esta é uma técnica amplamente utilizada.

No parágrafo anterior viu-se peculiaridades do projeto de interfaces. Infelizmente as dificuldades não existem só nesta fase. Conforme [Mye89], o software da interface é freqüentemente grande, complexo, difícil de depurar e modificar. Em [MR92] vê-se que projeto e implementação de interfaces ainda apresentam muitas questões para serem resolvidas satisfatoriamente. O software, conforme [MR92], supera em complexidade outros tipos de software e parece ser um problema de solução não próxima! Ainda são identificados três problemas técnicos no código de “boas” interfaces [SV91]: concorrência (através de diálogo *multithread*, veja §5.4.2), realimentação (§3.5.2) e dependências entre múltiplas visões.

Muitas pesquisas, contudo, têm abordado a construção de interfaces com o intuito de reduzir os custos de desenvolvimento e melhorar a qualidade das interfaces produzidas. Grande parte delas busca a solução através do emprego de ferramentas.

O objetivo destas ferramentas é fornecer suporte às etapas de desenvolvimento de interfaces: projeto, implementação, avaliação e manutenção.<sup>7</sup> Geralmente tem-se uma ferramenta orientada para cada atividade distinta de desenvolvimento. Quando as ferramentas estão integradas, o conjunto pode ser denominado de UIMS (*User Interface Management System*) (§3.6), UIDS (*User Interface Development System*) ou ainda UIDE (*User Interface Development Environment*).

Uma abordagem comum na construção de ferramentas é criar uma técnica de descrição ou linguagem de alto nível (suportada pela ferramenta) para descrever a atividade de interesse. Por exemplo, uma ferramenta que permite a geração automática de uma interface deve receber como entrada especificações da apresentação e do controle da interface e produzir como saída o código que implementa a interface especificada.

Estas especificações, contudo, ainda são problemáticas. A seção §3.5.7, por exemplo, mostra uma série de dificuldades para descrever o controle do diálogo de interfaces. Têm-se, entre elas: (1) a inabilidade de representar aspectos correlatos; (2) são (especificações) difíceis de usar e (3) muitas vezes não contemplam todos os tipos de interface.

Por último, convém ressaltar que o projeto de interface é complexo e difícil.<sup>8</sup> A construção iterativa exige protótipos para teste com o usuário. O código correspondente à interface em um sistema interativo é volumoso, complexo, e sua construção não é trivial. Ferramentas surgiram para automatizar o processo de construção de interfaces, contudo, ainda precisam

<sup>7</sup>Detalhes sobre o ciclo de vida de uma interface estão em §3.5.7.

<sup>8</sup>Exemplo: a apresentação de uma interface pode influir na confiabilidade do sistema. Como? Se duas operações de efeitos opostos são selecionadas com um simples clique no mouse e suas opções encontram-se próximas uma da outra, isto fatalmente provocará erros de operação. Assim, o projeto de interfaces envolve questões sutis que extrapolam o âmbito restrito do que se define como interface.



ser melhoradas, concentram-se em apenas alguns aspectos, não são fáceis de usar e as técnicas utilizadas para descrição apresentam restrições.

---

---

**Problema:** as técnicas atualmente empregadas para descrever o controle de interfaces apresentam inconvenientes.

---

---

### 1.3 Este trabalho

O problema contemplado neste trabalho são as dificuldades com o uso de técnicas para especificação do controle de diálogo. Há indícios (§1.5) favoráveis ao uso de Estadogramas<sup>9</sup> para especificação deste controle. Eles sugerem uma experimentação que permita identificar alterações promissoras que explorem o uso destes diagramas neste contexto específico. Esta experimentação deve envolver não só a notação (sintaxe), mas a semântica e a implementação desta notação.

O método empregado para obter tais alterações inclui o desenvolvimento de uma interface (§4.3) não-trivial<sup>10</sup> para servir de “bancada.” Neste desenvolvimento o controle de diálogo é registrado usando-se os Estadogramas. Para apoiar o desenvolvimento é utilizada uma ferramenta que automatiza a geração de código a partir de especificações em Estadogramas. Para os inconvenientes identificados durante o desenvolvimento são sugeridas soluções. Algumas são incorporadas na notação dos Estadogramas (sintaxe e semântica) e implementadas na ferramenta.

Este trabalho revela uma ênfase necessária[MR92] e que tem sido perseguida[RXW92]: síntese automática de interfaces, embora seja uma das atividades que mais tem recebido atenção dos estudiosos[OGL<sup>+</sup>87]. A observação empírica que este trabalho propõe também é empregada em outros trabalhos. O modelo estrela[HH89b] para o ciclo de vida de interfaces (§3.5.7), por exemplo, foi obtido por processo similar.

---

---

**Trabalho:** usa Estadogramas para especificação e implementação de controle de diálogo de interfaces e, simultaneamente, propõe “ajustes” para eliminar dificuldades encontradas neste emprego particular.

---

---

---

<sup>9</sup> Estadogramas são vistos em detalhes no capítulo 2. Trata-se de uma extensão aos diagramas de transição de estados convencionais. Por todo este texto usa-se Estadogramas ora como notação ora como linguagem, ou ainda Estadograma para representar a especificação.

<sup>10</sup> Aquela que apresenta dificuldades à construção. Por exemplo, possui diálogo concorrente, dependências entre múltiplas apresentações de um mesmo objeto e manipulação direta[SV91].

## Estabelecendo fronteiras

A abrangência deste trabalho já foi limitada à especificação e implementação de diálogo de interfaces. Existem, contudo, outros problemas nesta área multidisciplinar. Esta seção fornece uma fronteira para o escopo deste trabalho e elucida o que não é de interesse. Parte-se do âmbito de interfaces que é restringido paulatinamente, até a obtenção da área de atuação deste trabalho. Controle de diálogo é um termo fundamental e é visto na seção seguinte.

Pode-se identificar dois grandes contribuintes para a pesquisa em interfaces: ciência da computação e psicologia[Abo91].

Os fatores pertinentes à psicologia estão fora do presente escopo. Por exemplo, a divisão da tela em janelas que apresentam a razão áurea,<sup>11</sup> defendida por Gait[Gai86] são ou não mais agradáveis que as demais? Os menus *pie*[Hop91] são mais eficientes que os tradicionais? Metáforas são elementos capazes de melhorar a qualidade de interfaces? O mais importante é a sintaxe de uma linguagem de comandos ou a compatibilidade com o a língua nativa do usuário? Estas questões estão além dos objetivos deste trabalho. Estes fatores não são negligenciados, são de interesse do projeto de interfaces, que não é interesse deste trabalho.

O outro contribuinte, a ciência da computação, fica responsável por fornecer meios para apoiar o projetista desde a representação de suas idéias até a implementação em um computador. Deve haver uma nítida separação entre os papéis e resultados produzidos por cada área. Um especialista em computação, por exemplo, entende de computadores e, portanto, deve restringir-se a este domínio, seja pelo caráter multidisciplinar de interfaces ou pela falácia da intuição egocêntrica.<sup>12</sup> Em interfaces a ciência da computação deve ser vista como meio através do qual os *insights* da psicologia, a ciência que mais conhece o usuário, são utilizados.

Esta tese concentra-se nos aspectos computacionais de uma interface. Ainda mais pontual, interessa-se por uma notação utilizada para registrar um dos resultados do projeto: o comportamento (controle) da interface; e sua posterior implementação.

A especificação do comportamento da interface é uma atividade de projeto. Pode-se empregar Estadogramas nesta fase (para uso de um projetista de interface) ou na fase de implementação (para uso de um programador). Este trabalho usa Estadogramas como notação para registrar o comportamento da interface (atividade de projeto). Entretanto, isto precisa ser considerado futuramente, pois uma descrição em Estadogramas pode ser muito complexa para um especialista versado em fatores humanos.<sup>13</sup>

---

<sup>11</sup>A razão áurea é muito empregada, por exemplo, nas dimensões de livros de forma que suas proporções mantêm esta razão. O valor desta razão é o limite de seqüências de razões de números sucessivos de Fibonacci (1,2,3,5,8,13,...). Corresponde à aproximadamente 1,61803.

<sup>12</sup>Refere-se à ilusão de que se conhece o que determina o comportamento e satisfação do usuário. Conforme [Lan88], mesmo quando um sistema é obviamente "difícil" a intuição pode não revelar as verdadeiras razões. O melhor é, sempre que possível, permitir que o especialista adequado desenvolva a atividade que lhe compete.

<sup>13</sup>Um único indivíduo, geralmente o programador, pode realizar todas as atividades do ciclo de vida de uma interface. Este fato, entretanto, não reflete uma situação ideal.

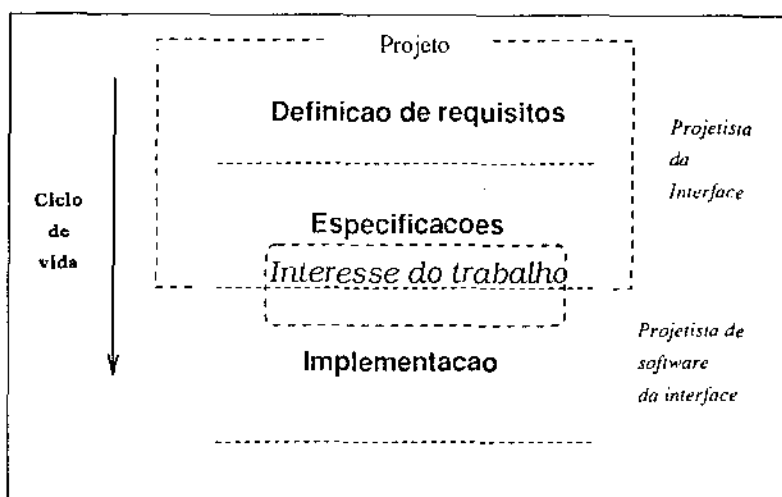


Figura 1.2: Área de interesse deste trabalho.

A figura 1.2 mostra, grosseiramente, a “região” de maior interesse deste trabalho no ciclo de vida de uma interface (§3.5.7). Do projeto interessa a especificação do diálogo realizada pelo projetista da interface. A área abaixo da linha<sup>14</sup> que separa a fase de projeto da implementação corresponde à preocupação com a atividade que deriva código para a interface. Ou seja, interessa a obtenção de código a partir da especificação do controle de diálogo. Esta área ainda contempla outras questões de implementação de interfaces (§5.4) que também foram consideradas.

Quando se diz interface espera-se que o leitor lembre-se de *WIMP*,<sup>15</sup> ou seja, interfaces que utilizam objetos comuns de interação. Outras interfaces não comuns envolvem comandos acionados por voz, multimídia<sup>16</sup>, *touch screen*, *pen computing*, *groupware*<sup>17</sup> e realidade virtual, por exemplo. Estas últimas geralmente envolvem novos meios de interação e possuem seus próprios problemas — são ditas *non-WIMP* e não foram consideradas.

### Especificação do controle de diálogo

Esta seção identifica o aspecto de uma interface de interesse neste trabalho: o controle de diálogo.

<sup>14</sup>O desenvolvimento de interfaces não apresenta divisão que possa separar fases ao longo do tempo, pois não se trata de processo linear (§3.5.7). A intenção desta divisão é ressaltar a distinção entre os objetivos destas fases.

<sup>15</sup>*Windows, Icons, Menus e Pointing device*.

<sup>16</sup>Do inglês *multimedia*. O uso de multimídia é desejável pela combinação de várias modalidades de comunicação. Conforme [Mvd91] pessoas têm facilidade de lembrar se há uma combinação de fazer, ouvir e ver.

<sup>17</sup>Software que permite vários usuários colaborarem com a realização de uma mesma tarefa.

O modelo de Seeheim (§3.3), esboçado na figura 3.3 (pág. 49), identifica claramente o controle de diálogo. O modelo de Seeheim é lógico, pois não diz como uma interface deve ser estruturada ou implementada, mas distingue os componentes lógicos que devem compreender uma interface. Neste modelo o módulo “controle do diálogo” (centro), responsável pelo controle da seqüência de símbolos de entrada e saída (sintaxe), corresponde ao módulo cuja especificação e implementação é de interesse neste trabalho.

Observando a figura 3.3 percebe-se que descrever uma interface não se resume à descrição do controle, há outros aspectos a serem descritos. O mais palpável seria a “apresentação,” encarregada de tornar visível o resultado do processamento e receber as entradas do usuário. Uma arquitetura para o software de uma interface é comentada em §1.2.

## 1.4 Objetivos

Muitos trabalhos na área de controle de diálogo de interfaces preocupam-se somente com a descrição do controle, os demais aspectos são descartados. Outros como [Wel89] empregam Estadogramas, ou outra notação mas carecem de exemplos substanciais. Ainda há aqueles que se voltam para questões específicas de implementação. Enfim, a maioria dos trabalhos concentra-se em seus aspectos de interesse e deixa, muitas vezes, dúvidas (Como? Quando? Em que situações empregar?). Só um trabalho abrangente e o espaço de um livro poderiam conter uma visão abrangente e didática destas questões. Isto dificulta, em parte, uma visão geral de como se processa o desenvolvimento de uma interface. Este trabalho não tem a pretensão de preencher esta lacuna, mas busca fornecer informações que permitam visualizar um panorama para o emaranhado de questões que envolvem o desenvolvimento de uma interface.

---

**Objetivo:** identificar dificuldades e vantagens do emprego de Estadogramas para especificação de interfaces; propor, na medida do possível, soluções para amenizar os problemas e fornecer um panorama da aplicabilidade desta notação neste contexto.

---

Não se deseja avaliar o uso de Estadogramas, mas apenas registrá-lo juntamente com as considerações julgadas oportunas para o presente emprego. Naturalmente este trabalho não sela o destino dos Estadogramas neste contexto. São obtidos outros resultados:

1. dificuldades identificadas com a notação e
2. restrições que a implementação desta notação deve satisfazer para se adequar ao software de uma interface.

Os motivos e justificativas para execução deste trabalho seguem adiante (§1.5). Ainda

considere que o investimento em uma linguagem (compilador, interpretador, depurador (*debugger*), otimização) não pode ser desprezado. É preciso ter informações mínimas que justifiquem a construção destas ferramentas para uma linguagem. Este trabalho fornece indicadores que auxiliam esta tomada de decisão para os Estadogramas neste contexto específico.

### Algumas considerações

Embora além do escopo deste trabalho, o desenvolvimento da interface permite contato com as questões abaixo e, de certa forma, são estímulos para a execução deste trabalho:

- O desenvolvimento de sistemas reativos (§2.1) ainda não possui métodos satisfatórios que possam ser empregados[HLN<sup>+</sup>90]. Estadogramas é uma proposta para registrá-los. Inexistem “regras de ouro” para o uso desta notação. Drusinsky e Harel[DH89] afirmam que esta questão requer investigações. Faltam exemplos substanciais de emprego em situações e sistemas distintos. Serão as observações dos resultados destes trabalhos empíricos que irão colaborar na identificação de diretrizes.
- Métodos tradicionais (voltados para dados) são inadequados para o desenvolvimento de sistemas interativos[WPSK86]. Existem muitas notações e ferramentas para projeto e implementação de interfaces (§3.6). Tais notações e ferramentas, contudo, estão isolados dos métodos de desenvolvimento tradicionais, essencialmente voltados para aspectos funcionais[SB89]. Em [HH89b] é destacada a necessidade de uma metodologia de desenvolvimento unificada, que contemple o desenvolvimento de interface como parte do processo de engenharia de software.

O desenvolvimento da interface neste trabalho faz uso do conceito de independência de diálogo (pág. 52) e seguiu, tanto quanto possível, isolado da aplicação. O ambiente Statemate[HLN<sup>+</sup>90] (§2.9) apresenta uma metodologia que emprega Estadogramas para desenvolvimento de sistemas complexos. A questão principal talvez seja identificar “quando usar” e quais os “efeitos colaterais” do emprego desta notação no desenvolvimento de sistemas interativos.

- A descrição do desenvolvimento também é útil para estudiosos de interação entre homem e computador. Conforme Landauer[Lan88], um dos melhores métodos para fazer pesquisa sobre interação homem-computador envolve a manipulação de projetos de sistemas e observação das conseqüências.

## 1.5 Motivação

Os itens abaixo justificam e fornecem motivações para a execução deste trabalho e o emprego de Estadogramas para especificação de diálogo. Referem-se, num nível mais elevado, a dois pontos primordiais:

*As técnicas atualmente empregadas para especificação de controle de diálogo apresentam problemas (§3.5.7).*

*Estadogramas possuem indícios de que se aplicam na solução de alguns desses problemas. Se se desconhece um trabalho que fornece uma perspectiva deste emprego, por que não tentar obtê-la?*

Justificativas:

- O comportamento de um sistema reativo (§2.1) é difícil de ser descrito de forma clara e precisa[Har87]. No âmbito de desenvolvimento de interfaces vários formalismos já foram utilizados para descrever esse comportamento, no entanto, cada um tem seus próprios inconvenientes. Nenhuma notação obtém consenso na literatura e, portanto, *a especificação da interação do usuário com o computador ainda não é assunto definido*. Em [BC91, pág. 165-167] são apresentadas dificuldades com modelos usados para descrever diálogos. Estadogramas chegam a ser citados como extensão de máquinas de estados finitos, mas nenhum outro comentário é feito.
- *Estadogramas possuem capacidade de especificação de diálogo igualável a outros modelos (§2.7) além de maior expressividade em alguns casos*. Estende os diagramas de transição de estados (DTEs) que caracterizam, em parte, os UIMS's<sup>18</sup> de segunda geração[Hix90]. Embora estes diagramas apresentem problemas conhecidos, continuam em uso. Em [HH89b] é comentada uma recente extensão destes diagramas.
- Estadogramas foram propostos recentemente tendo em vista as deficiências encontradas em outras notações para a descrição de sistemas reativos. *As qualidades e deficiências dos Estadogramas precisam ser identificadas para a especificação do diálogo de interfaces*, principalmente diante dos indícios favoráveis. Isto só pode ser obtido através da análise de emprego desta notação em projetos reais, ou seja, trabalho similar ao realizado em [DH89]. Gehani e MacGetrich[GG86] exaltam a necessidade da observação da aplicação de linguagens de especificação a problemas reais e substanciais como elemento imprescindível em discussões sobre linguagens de especificação.
- Existem ferramentas que permitem a construção de protótipos e a redução do complexo e volumoso software de interface (§3.6). *Este trabalho ajuda a identificar o suporte que está faltando a esta notação para construir uma ferramenta de qualidade*. Só quando tal ambiente existir poderão ser feitas comparações mais realísticas entre Estadogramas e outras notações para especificação de controle de diálogo.

---

<sup>18</sup>Acrônimo de *User Interface Management System* (§3.6).

## 1.6 Visão geral da dissertação

No capítulo 2 uma compilação acerca de Estadogramas fornece uma visão geral deste formalismo e introduz sua notação. Fornecida a ferramenta de especificação é necessário identificar o contexto em que é empregada. Por conseguinte, metodologias, técnicas de desenvolvimento, princípios de projeto e modelos de interação entre o homem e o computador são apresentadas no capítulo 3. Este capítulo serve para orientar o desenvolvimento proposto, identificar as dificuldades e mostra como foram resolvidas, e permite localizar o papel dos Estadogramas no desenvolvimento de interfaces. O capítulo 4 descreve o sistema para o qual a interface é construída. Neste capítulo ainda consta o processo de desenvolvimento da interface. A interface desenvolvida como parte do trabalho valida a especificação, que é analisada no capítulo 5. Este capítulo ainda traz uma série de alterações propostas para facilitar a especificação de interfaces. Na conclusão são resumidos os problemas, as soluções adotadas e a experiência do desenvolvimento empírico.

### Resumo dos capítulos

**Capítulo 2** Fornece uma introdução aos sistemas reativos e uma visão geral dos Estadogramas (descrição, empregos e ferramentas que fornecem suporte). Também compara informalmente a capacidade de expressão dos Estadogramas com outras notações.

**Capítulo 3** Discorre sobre interação homem-computador em vários aspectos. São apresentados conceitos comumente usados, modelos, arquiteturas, projeto e implementação de interfaces. O capítulo reflete as atuais tendências nesta área. Fica caracterizada a complexidade do desenvolvimento de interfaces e justificada a alta demanda de tempo para construí-las.

**Capítulo 4** Descreve o sistema para o qual uma interface é construída; a interface e resumidamente o desenvolvimento. Ainda inclui descrição do ambiente de desenvolvimento. Para a descrição da interface são apresentados alguns cenários da interação com a mesma.

**Capítulo 5** Descreve alterações consideradas oportunas para o presente emprego dos Estadogramas. As alterações são obtidas do desenvolvimento da interface (descrita no capítulo anterior) e parte da observação de referências na literatura sobre o assunto. Ainda discorre sobre questões relacionadas à implementação dos Estadogramas.

**Capítulo 6** Resume o trabalho realizado, descreve conclusões, trabalhos futuros e considerações finais.

## 1.7 Convenções

1. Para facilitar a leitura, alguns termos foram simplesmente transcritos da língua de origem. Por exemplo, usa-se hardware, software sem qualquer distinção das palavras da nossa língua. Exemplo de outros termos empregados: mouse e menu. Alguns termos traduzidos são seguidos do original em itálico.
2. O símbolo § precede um identificador que designa capítulo ou seção.
3. Preferiu-se termos aportuguesados como leiaute a *layout*. Os primeiros são encontrados no Novo Dicionário da Língua Portuguesa (Segunda edição — revista e ampliada), Aurélio Buarque de Holanda, Editora Nova Fronteira.

## 1.8 Resumo

Reconhece-se a relevância das interfaces e as dificuldades enfrentadas no processo de suas construções. Na fase de projeto um dos aspectos a serem descritos é o comportamento, que corresponde à sintaxe da interação do usuário com um sistema. A descrição desta sintaxe também é problemática e várias técnicas existem, suportadas por ferramentas, para registrá-la. Esta dificuldade é um dos estímulos citados para o presente trabalho. Restringiu-se a consideração de questões psicológicas das interfaces. Interessa-se por software para interfaces, especialmente a sua produção. Para isto emprega-se Estadogramas no desenvolvimento de uma interface não-trivial; são sugeridas mudanças na notação e realizadas considerações acerca da implementação destes diagramas.

Uma visão geral deste trabalho foi descrita neste capítulo, as motivações, justificativas e objetivos. No próximo capítulo são fornecidos detalhes sobre os Estadogramas, um promissora notação!



## Capítulo 2

# Estadogramas

*“Boas especificações são a chave para o desenvolvimento de software.”*  
Neumann[Neu89]

No capítulo anterior vimos uma visão geral deste trabalho que propõe, entre outras atividades, a implementação de uma interface usando Estadogramas como notação para registrar o diálogo. Neste capítulo são apresentadas duas perspectivas dos Estadogramas: a notação (sintaxe) e a implementação (semântica). As descrições realizadas são informais.

A sintaxe representa os elementos existentes neste formalismo para a especificação de sistemas reativos. Limitam-se a um subconjunto presente na corrente implementação.

A implementação é a realização dos Estadogramas. Trata-se da interpretação e execução da sintaxe. Há outras considerações acerca da implementação: a sua geração automática e o seu relacionamento com a arquitetura de software geralmente usada para interfaces.

Foram propostas mudanças consideradas oportunas que afetam a notação e a implementação. Algumas alterações são descritas neste capítulo por se relacionarem intimamente com os sistemas reativos, as demais encontram-se no capítulo 5 e dirigidas para o controle de interfaces. Ainda é fornecido um breve histórico: origem, empregos e ferramentas que usam Estadogramas.

---

### 2.1 Origem

Estadograma surgiram para descrever sistemas reativos, devido à insatisfação com técnicas até então empregadas. Esta seção define os sistemas reativos, identifica o problema de tais sistemas (comportamento complexo, que é difícil de ser descrito), e fornece uma taxonomia para eles.



Figura 2.1: Sistema transformacional.

### Uma nova dicotomia: sistemas transformacionais/reativos

Existem dicotomias aplicadas a sistemas (não exclusivamente de software) que separam os sistemas “difíceis” ou “problemáticos” dos demais. Sistemas síncronos/assíncronos, *off-line/on-line*, e seqüenciais/concorrentes são alguns exemplos. Respectivamente, os sistemas assíncronos, *on-line* e concorrentes seriam aqueles que necessitam de tratamento especial, assim como os não-determinísticos. Em contrapartida, os sistemas determinísticos, *off-line*, seqüenciais, fornecem menos dificuldades que seus complementares. Em [HP85] é apresentada outra dicotomia — sistema reativo/transformacional.<sup>1</sup>

---

**Sistema transformacional (voltado para dados):** recebe alguma entrada, realiza transformações sobre ela e produz a saída. A saída pode ser descrita como resultado de uma função aplicada à entrada. Ex.: sistemas para confecção de folhas de pagamento, previsão meteorológica, contabilidade e outros.

---

Um sistema transformacional (figura 2.1) obtém a saída  $S$  em função (transformação) da entrada  $E$ . Este sistema poderia, por exemplo, ser encarregado de previsão meteorológica (*number crunching*), onde  $E$  compreenderia todas as informações necessárias para a previsão e  $S$  seria a saída obtida após o processamento da entrada.

---

<sup>1</sup>Do inglês *transformational*. Faça-se distinção de outra acepção atribuída a transformacional — sistema obtido por sucessivas transformações que conservam uma equivalência funcional entre elas. Aqui transformacional qualifica o sistema e não o método de sua construção.

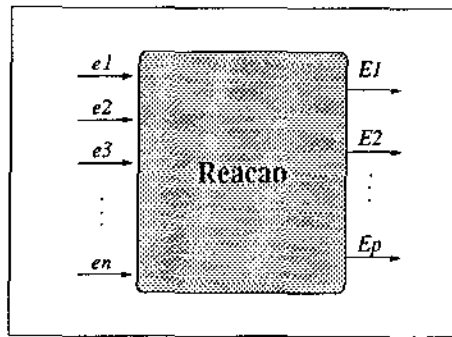


Figura 2.2: Sistema reativo.

---

**Sistema reativo** (*controlado por eventos*): continuamente responde (reage) a estímulos externos e internos. Em geral não computa ou executa uma função, mas mantém ininterrupto relacionamento com o ambiente. A saída é a reação à entrada. Ex.: sistemas de comunicação, interfaces homem-computador, sistemas embutidos,<sup>2</sup> software de tempo real e outros. Um relógio, caixa automático (banco) e televisão também são exemplos. São sistemas problemáticos<sup>3</sup> devido à carência de métodos adequados para a descrição dos seus comportamentos complexos.

---

Quando se usa o termo sistema reativo ele deve ser entendido como o sistema cujo principal, não o único, componente é o núcleo reativo (§2.3).

Um sistema reativo (figura 2.2) produz eventos de saída ( $E_i$ ) conforme a seqüência de eventos de entrada ( $e_j$ ) anteriormente obtida. A especificação do comportamento de um sistema reativo compreende um conjunto de seqüências legais de eventos de entrada e saída.

Por exemplo, um caixa automático continuamente espera por eventos provocados pelos clientes de um banco. Um cliente primeiro escolhe (provoca um evento) uma operação entre uma série de opções (extrato, saldo, ...), como reação à escolha o sistema pergunta o número da conta ao cliente. Se o número existir (condição) pergunta a senha e assim por diante. Em qualquer momento o cliente pode interromper e reiniciar o processo, i.e., gerar um evento que provoca um retorno ao estado inicial. Se o cliente demorar demasiadamente para fornecer qualquer informação o processo é interrompido e o sistema retorna ao estado inicial, ou seja, um intervalo de tempo é suficiente para provocar um evento. Vê-se que se trata de um sistema controlado por eventos. **Reagir** resume a principal função destes sistemas.

<sup>3</sup>Do inglês *embedded*. Sistemas em que o software é um de muitos componentes e freqüentemente não apresenta interface com o usuário, sua interface é com dispositivos. Geralmente controla outros componentes.

<sup>2</sup>Entenda como sistemas que requerem métodos e abordagens especiais para desenvolvimento.

## 2.2 Qual o problema com sistemas reativos?

Alguns sistemas precisam de linguagens apropriadas para descrição do comportamento complexo que apresentam. Um caixa automático (exemplo da seção anterior), embora de comportamento relativamente simples, oferece dificuldades quando se tenta especificá-lo informalmente em linguagem natural. O exemplo da seção anterior omitiu muitas considerações e, naturalmente, é incompreensível para quem não conhece um caixa automático. Ainda há casos bem mais complexos que um caixa automático. Uma linguagem informal ou imprecisa é de pouca utilidade nestes casos. O início do capítulo 5 discute um pouco mais sobre especificações formais e informais.

Os sistemas reativos apresentam um comportamento complexo, são recheados de eventos condicionais, a noção de contexto (estado) está presente e muitas vezes conjuntos de estados encontram-se ativos simultaneamente refletindo uma concorrência existente no sistema modelado.

Uma característica comum aos sistemas reativos é o seu comportamento complexo, que não é devidamente descrito por um simples relacionamento que especifica a saída como função da entrada, mas requer a relação entre saída e entrada através de combinações permitidas ao longo do tempo. Conforme Harel[HP85], métodos existentes para o desenvolvimento de sistemas passo a passo e bem-estruturados são predominantemente transformacionais. Os diagramas de estrutura<sup>4</sup> são suficientes para descrever o comportamento de sistemas transformacionais, que também beneficiam-se da decomposição funcional. Tal decomposição, contudo, não se aplica ao comportamento de sistemas reativos. É interessante observar o reflexo desta afirmação no âmbito de interfaces (exemplo de sistema reativo). Draper[DN85] faz a mesma afirmação destacando a aplicação imprópria de métodos tradicionais no desenvolvimento de interfaces.

*Os métodos geralmente empregados para descrição de sistemas transformacionais são inconvenientes para a descrição de sistemas reativos, e os métodos existentes atualmente para esta finalidade apresentam dificuldades[Har87].* Em conseqüência, Estadogramas são propostos em [Har87]. Existem várias linguagens para a especificação de sistemas reativos. Cada uma delas, naturalmente, apresenta qualidades e inconvenientes. Estadogramas tentam reunir boas qualidades (ecclético) e eliminar inconvenientes das linguagens existentes para este propósito. Várias ferramentas (§2.9) apóiam a construção de sistemas reativos, inclusive os sistemas interativos.

---

<sup>4</sup>Diagramas usados pela análise estruturada de sistemas para descrever fluxo de controle.

## 2.3 Uma taxonomia para sistemas reativos

Conforme Berry e Gonthier[BG92] um sistema reativo pode ser visto como uma composição de três camadas:

- Uma **interface** com o ambiente responsável pelo recebimento das entradas e produção da saída. Transforma eventos físicos (externos) em lógicos (internos) e vice-versa. O núcleo reativo manipula eventos lógicos. A interface comunica-se com o núcleo reativo através de um protocolo.
- Um **núcleo reativo** que contém a lógica ou comportamento do sistema. Gerencia as entradas e saídas lógicas. Decide qual a reação a ser tomada em resposta à entrada. O Tradutor Blob (§2.9.4) e um *run-time* permitem a geração automática desta camada a partir de especificações em uma linguagem de alto nível para descrição textual dos Estadogramas.
- **Aplicação** (*data handling*) é a camada responsável pela funcionalidade do sistema, i.e., as transformações de dados.

O núcleo reativo é a camada de maior interesse deste trabalho. Sua especificação e implementação são fundamentais, pois trata-se da camada de maior complexidade e importância dos sistemas reativos. O núcleo reativo também qualifica um sistema reativo, i.e., diz-se que um sistema é reativo quando o núcleo reativo é o seu principal componente. A seção seguinte relaciona cada uma destas camadas com a arquitetura de software geralmente empregada para interfaces.

## 2.4 Relação entre sistemas reativos e modelos de interface

A figura 2.3 mostra a relação entre modelos de interfaces e a taxonomia fornecida na seção anterior. Na horizontal tem-se os sistemas reativos, por exemplo, a segunda linha exemplifica a taxonomia fornecida anteriormente neste capítulo. Esta taxonomia é um modelo de sistema reativo. As demais linhas são modelos e arquiteturas de software para interfaces. Na vertical tem-se as divisões em componentes propostas pelos modelos para sistemas reativos e interfaces.

Os modelos e arquiteturas são comentados no capítulo 3. A relação também mostra o emprego de vários termos com o mesmo significado. De cima para baixo tem-se: a taxonomia (seção anterior); o modelo de Seelheim e o modelo de protocolo virtual (modelos são vistos em §3.3), e uma divisão em camadas para o software de sistemas interativos (§3.4).

O comportamento de uma interface é ditado pelo diálogo ou sintaxe, visto aqui como o núcleo reativo (parte de um sistema reativo). A interface de um sistema reativo (§2.3) possui as mesmas finalidades do componente léxico de uma interface homem-computador (§3.3).

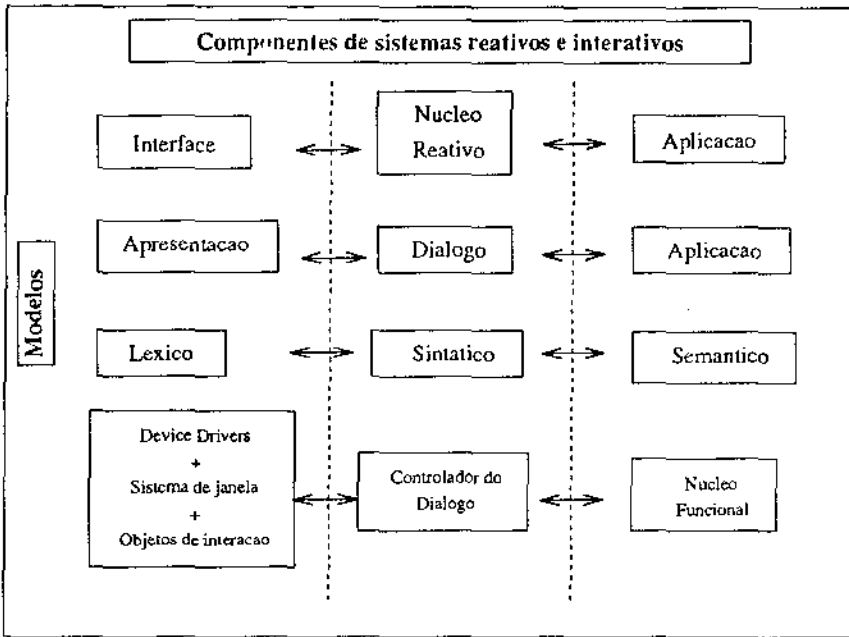


Figura 2.3: Relacionando modelos.

Relacionando-o com a arquitetura de software em §3.4, o componente léxico é encarregado das funções dos objetos de interação.

A aplicação ou núcleo funcional (§3.4) equivale à a camada de aplicação (§2.3).

Pode-se questionar a existência de funções de uma interface que não podem ser descritas usando-se Estadogramas, ou seja, pensando-se apenas no comportamento. O componente léxico é responsável pela apresentação e não é de construção trivial! No entanto, este trabalho considera como sistema reativo o sistema cujo principal componente é o seu comportamento. O gargalo no desenvolvimento destes sistemas é estabelecer as saídas enviadas ao ambiente externo como respostas às entradas recebidas.

Este trabalho utiliza Estadogramas para a descrição do comportamento de uma interface, ou seja, a coluna central da figura 2.3.

## 2.5 O que são Estadogramas?

Vimos uma classe de sistemas considerados complexos com relação ao comportamento: os sistemas reativos. Esta seção define sistemas reativos, descreve a notação dos Estadogramas e parte das adaptações propostas para uso na especificação do controle de diálogo.

---

---

**Estadogramas:** diagramas propostos por Harel[Har87] para descrição do comportamento de sistemas reativos. Estende os diagramas de transição de estados eliminando alguns inconvenientes e retendo a natureza gráfica dos últimos. As extensões acrescentam concorrência, comunicação (*broadcast*) e hierarquia.

---

---

Harel em [Har88] enfatiza duas características importantes dos Estadogramas:

- São **visuais**, pois podem ser representados graficamente. Isto facilita a sua compreensão.
- São **formais**, em conseqüência, permitem a manipulação, manutenção e análise automática.

À semelhança dos diagramas de transição de estados, Estadogramas são baseados em estados, eventos e condições. Eventos e condições podem ser combinados para causar transições entre estados. Ações são as respostas dos Estadogramas à transições. Ações podem controlar atividades (fornecem semântica à aplicação). Portanto, a entrada correspondente a uma especificação em Estadograma compreende estímulos (eventos) externos e aqueles gerados internamente, enquanto a saída são ações. A união das saídas e entradas permitidas ao longo do tempo forma a interface do controle de um sistema.

## Empregos

A notação dos Estadogramas apresenta qualidades para a especificação de hardware[DH89]. Em [DH89] também é levantada a hipótese do uso de Estadogramas como linguagem de programação para sistemas concorrentes.

Harel[Har87] sugere várias aplicações dos Estadogramas: especificação de diálogo de interfaces, descrição de hardware, protocolos de comunicação, sistemas de tempo real (*real-time*) e outras. Nesta dissertação, o software de um sistema interativo responsável pelo controle da interação é especificado usando-se Estadogramas. Este emprego não é inédito[Pin90], além de ser sugerido em [Har87]. Statemaster é um UIMS que usa Estadogramas exclusivamente para esta finalidade.

As extensões sobre os diagramas de transição de estados podem ser vistas em §2.7.1.

## 2.6 Descrição informal

Está além do escopo deste trabalho apresentar uma descrição completa dos Estadogramas e exemplos do seu uso. Uma descrição exaustiva é fornecida em [Har87]. Outras incluem

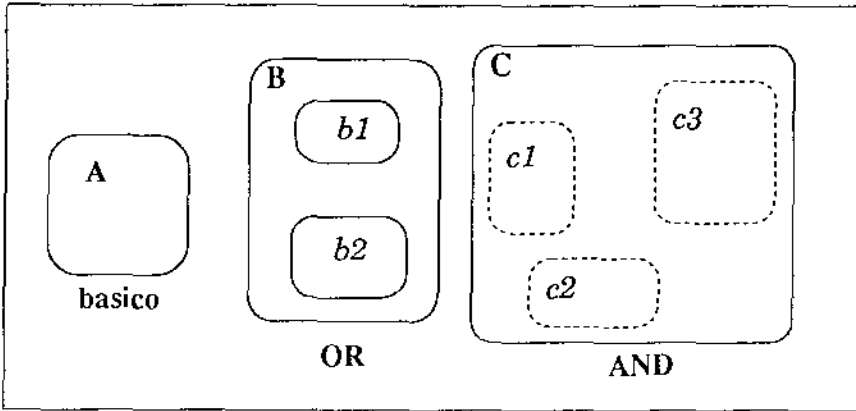


Figura 2.4: Tipos de estados em Estadograma.

[HP85] e [Har88]. A semântica formal pode ser obtida em [HPSS87, HGdR88, KP92, HRdR92] e [PS91]. Uma descrição informal pode ser encontrada em [iLI87] e [iLI89a]. Ambientes que empregam Estadogramas são descritos em §2.9.

A descrição dos Estadogramas segue um subconjunto e incorporar adaptações à especificação em [Har87]; isto porque o uso dos Estadogramas em determinadas áreas força alterações na linguagem “oficial” para se adequar à situações particulares. Algumas ferramentas (§2.9), p. ex. Reacto (§2.9.3), modificam a especificação feita por Harel. Harel em [Har87, pág. 256] também afirma que não possui uma recomendação final para a sintaxe nem para a semântica.

Este trabalho usa o Tradutor Blob (§2.9.4) e um *run-time* (§2.8) para implementação dos Estadogramas. As modificações e acréscimos que implicam em alterações em um destes elementos são comentadas no capítulo 5.

(NOTA. Neste trabalho existem modificações no nível léxico, sintático e semântico. As alterações encontram-se sublinhadas para facilitar a identificação.)

## Descrição

---

**Estado:** retângulo com cantos arredondados representa um estado. Um estado captura uma situação ou modo conceitual (veja pág. 84). Há três tipos de estados: “básico”, “OR” e “AND.”

---

Na figura 2.4 vemos vários estados identificados pelos rótulos A, B e C. O estado A é um estado básico, i.e., não possui subestado.<sup>5</sup> O estado B (tipo OR) contém dois subestados:

<sup>5</sup>Estadogramas são hierárquicos e diz-se que um estado  $s$  é subestado de  $E$  quando  $E$  for ancestral direto



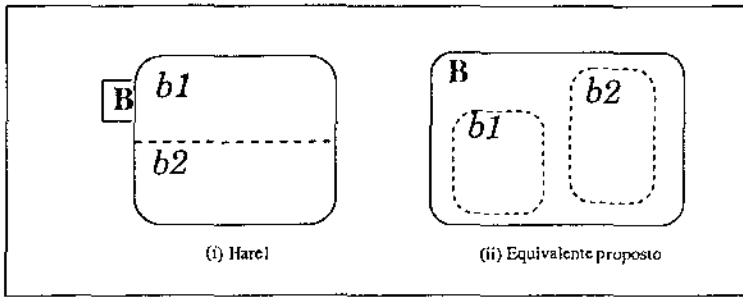


Figura 2.5: Estados concorrentes.

$b_1$  e  $b_2$ . “Estar<sup>6</sup>” no estado **B** significa estar em um de seus subestados, ou  $b_1$  ou  $b_2$ , nunca em ambos simultaneamente. Por outro lado, estar em **C** (estado tipo AND) significa estar em  $c_1$ ,  $c_2$  e  $c_3$  simultaneamente. Por conseguinte, os subestados de **C** são ditos concorrentes ou ortogonais, pois jamais o estado  $c_1$  será corrente sem que  $c_2$  e  $c_3$  também o sejam.

Estados ortogonais são representados por linhas tracejadas. Isto permite uniformidade na nomeação dos nós, o que não é possível com os estadogramas de Harel (originais), exceto se criarmos estados adicionais e desnecessários. Após esta alteração para representar o fato de que dois estados  $b_1$  e  $b_2$  são ortogonais, a linha de contorno destes estados deve ser tracejada. Nos Estadogramas de Harel uma linha tracejada separa estados concorrentes. A figura 2.5 mostra a versão de Harel (i) e a utilizada (ii).

A hierarquia não induz reações concorrentes. Por exemplo, na especificação original ao ocorrer o evento  $e$  (figura 2.6) duas ações concorrentes ocorreriam simultaneamente: a entrada no estado **A** e **B**. No presente emprego, o estado de nível mais elevado (estado **A**) tem prioridade sobre o estado de nível inferior (estado **B**) e impõe uma ordem de execução se houver reações concorrentes para serem executadas. Na figura 2.6, portanto, primeiro entra-se no estado **A** e só então no estado **B**, pois neste trabalho a hierarquia é vista como uma definição acumulativa de contextos.

Especificações em Estadogramas interagem com o “mundo exterior” através de eventos (entrada) e ações (saída).

ou indireto de  $s$  ao longo desta hierarquia.

<sup>6</sup>Este verbo é usado aqui para representar conceitualmente que o sistema modelado encontra-se em certa situação ou condição, representada por um estado. Também é usado “atingir o estado **E**” com a acepção de “tornar corrente o estado **E**.”

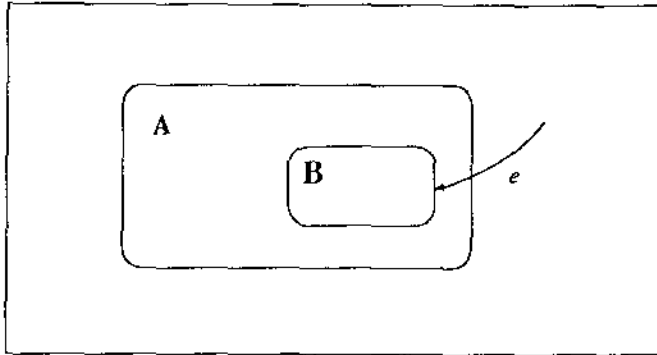


Figura 2.6: Estabelecendo prioridades.

---

**Evento:** corresponde a uma entrada fornecida a um Estadograma. Evento é a mola propulsora de Estadogramas. Este trabalho considera apenas sistemas de software, portanto, a entrada é através de eventos lógicos. Fisicamente correspondem a ações do usuário sobre dispositivos de entrada, podem ser gerados pelo software da aplicação ou por consequência de uma transição. Neste último caso, afetam componentes concorrentes.

---

---

**Atividade:** processamento responsável pela funcionalidade de um sistema. Pode representar uma resposta ao usuário ou um processamento longo (*number crunching*), por exemplo. São controladas por ações que podem aparecer como reação de uma transição ou na entrada ou saída de um estado.

---

Note que atividades não são descritas por uma especificação em Estadogramas. Relacionam-se com a parte funcional e são geralmente descritas usando-se diagramas de fluxos de dados.

---

**Ação:** mecanismo utilizado por Estadogramas para expressar saída para o "ambiente externo," gerar eventos capazes de afetar componentes ortogonais, definir o estado inicial de um contexto na ativação de um estado e efetuar operações de *housekeeping* por ocasião da desativação de um estado. Conceitualmente não consome tempo, o objetivo é preservar uma "reação em cadeia," considerando-a uma unidade indivisível.

---

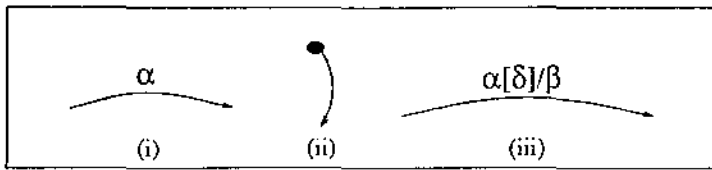


Figura 2.7: Arestas em Estadograma.

Ação é semelhante às saídas das máquinas de Mealy e Moore.<sup>7</sup> Ações são instantâneas e são utilizadas para controlar atividades (responsáveis pela funcionalidade do sistema modelado). Harel criou algumas ações específicas para controlar atividades: `start(X)` e `stop(X)`; e a condição `active(X)`. Na implementação corrente ações correspondem a chamadas de procedimento. Assim, procedimentos podem ser especificados para execução durante uma transição, ou quando se sai ou entra em um estado.

---

**Aresta:** mecanismo utilizado para estabelecer possíveis transições entre estados. Uma aresta é representada por um arco orientado que geralmente liga dois estados. Arestas podem ser rotuladas com uma combinação de eventos, condições e ações. Aresta é o único meio através do qual um estado pode ser atingido, explícita ou implicitamente.

---

A sintaxe do rótulo de uma aresta é  $\alpha[\delta]/\beta$ , onde  $\alpha$  é o evento necessário para que uma transição ocorra;  $\delta$  é uma condição que evita a transição caso seja falsa, e  $\beta$  é uma ação disparada quando ocorre a transição. Todos estes elementos de um rótulo são opcionais. Abaixo é fornecido o significado de alguns rótulos.

A figura 2.7 fornece exemplos de arestas. No exemplo (i), se o evento  $\alpha$  ocorre, a aresta é percorrida. Note que não existe condição que possa impedir a transição. Quando a “origem” de uma aresta é representada por um ponto hachurado não temos ligação entre estados e o rótulo deste tipo de aresta sempre é vazio (veja (ii)). Esta aresta é denominada aresta *default*. O estado destino é dito estado *default*. Em (iii), se ocorre  $\alpha$  e a condição  $\delta$  é verdadeira neste instante, ocorre a transição e a ação  $\beta$  é disparada.

Na figura 2.8 vemos os estados **A** e **B** ligados pela aresta rotulada apenas com o evento  $e$ . Neste caso, a ocorrência do evento  $e$  é a condição necessária e suficiente para causar a transição do estado **A** para o **B**. Note ainda que o estado *default* é o **A**, ou seja, ao simular o comportamento deste Estadograma o primeiro estado visitado é o estado **A**.

Uma reação pode ser causada por um evento sem ocorrer uma transição. Por exemplo, no ambiente *Windows* (apêndice A) uma aplicação pode, por um erro, escrever na tela em área que pertence a outra aplicação. Permitir que isto possa ser registrado necessita de algum

<sup>7</sup> *Theory of Finite Automata*, Prentice Hall, Johan Carrol e Darrell Long, 1989.

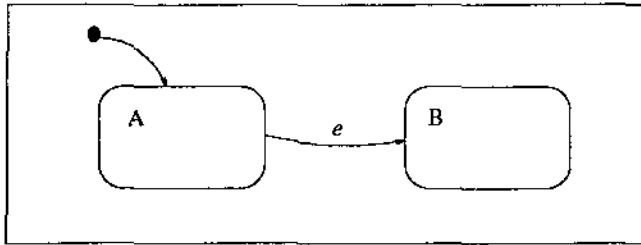


Figura 2.8: Transição entre estados.

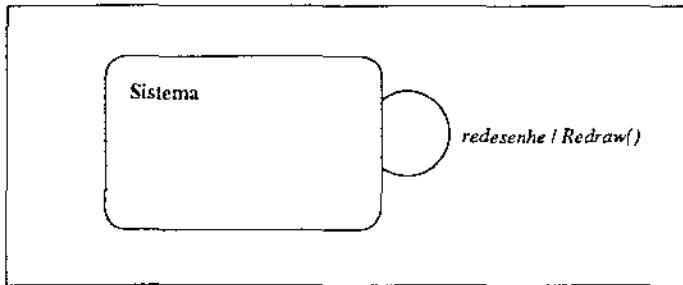


Figura 2.9: Aresta que não causa transição.

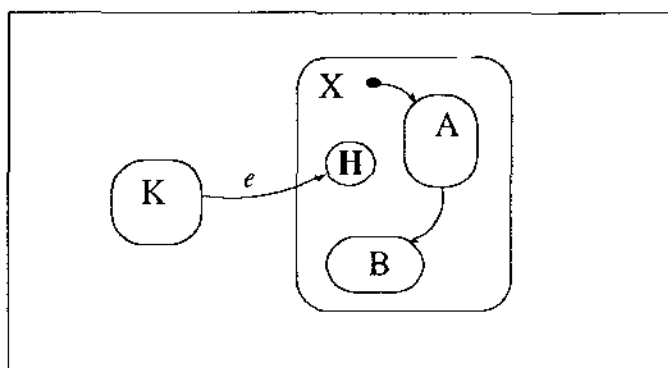
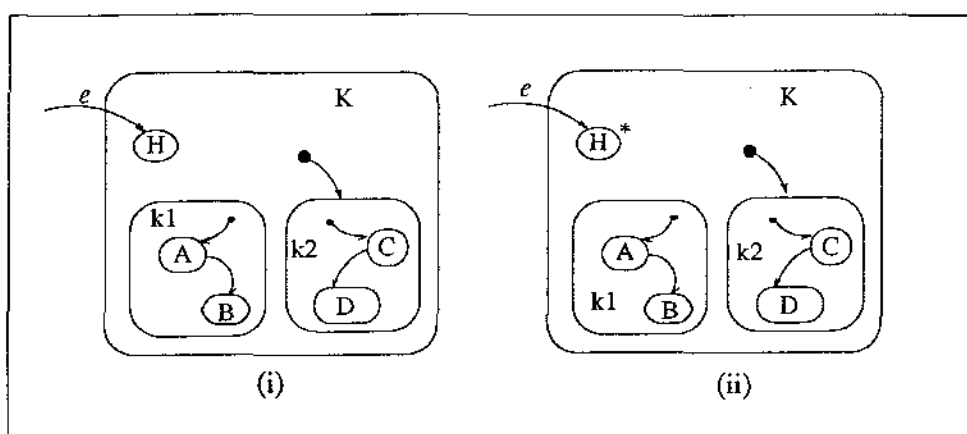
recurso para redesenhar toda a sua área e sem alterar o “estado” anterior ao seu uso. Este recurso é melhor modelado como uma ação a ser executada sempre que o evento correspondente for gerado. Não existe a necessidade de uma transição. Convém lembrar que os Estadogramas originais não geram uma ação se não existir uma transição. Para representar graficamente esta situação é proposta uma extensão conforme figura 2.9. A figura é interpretada da seguinte forma. Caso o estado *Sistema* esteja ativo sempre que ocorrer o evento *redesenhe*, a ação *Redraw()* é executada. Não ocorre uma suposta saída do estado *Sistema* seguida de uma entrada, não há transição. Isto é descrito sem ambigüidade através da aresta não orientada. A necessidade de modelar esta situação também é identificada por Wellner[Wei89]. Wellner, no entanto, não utiliza uma representação gráfica para identificar esta situação. A figura 4.7, na página 74, fornece outro exemplo de um caso desejável deste tipo de transição.

---

**History:** descreve modo alternativo de ativação de subestados. Entrada por *history* em um estado conduz ao subestado mais recentemente visitado. Se for *history* tipo  $H^*$  considera-se o estado mais recentemente visitado independente do nível. Caso contrário, considera-se apenas estados no mesmo nível da entrada e daí prossegue-se com arestas *default*.

---

Na figura 2.10 temos uma entrada no estado X por *history* simples, ou seja, ao ocorrer o

Figura 2.10: Exemplo de *History* simples.Figura 2.11: Exemplo de *History H\**.

evento  $e$ , caso o estado corrente seja o estado  $K$  o próximo estado será o mais recentemente visitado entre  $A$  e  $B$ . Caso seja a primeira vez que o estado  $X$  é visitado então segue-se a aresta *default* que conduz ao estado  $A$ . Note que este tipo mais simples de *history* aplica-se somente aos estados que se encontram no mesmo nível da entrada. Desse modo, se  $A$  e  $B$  não forem estados básicos, a entrada em  $X$  pelo evento  $e$  irá conduzir ao subestado *default* de  $A$  ou  $B$ . A figura 2.11 (i) exemplifica esta situação com outro exemplo. A ocorrência do evento  $e$  na figura 2.11 (i) irá conduzir ao subestado  $A$  ou  $C$  (estados *default* de  $K1$  e  $K2$ , respectivamente).

A figura 2.11 (ii) mostra outro tipo de *history*, o *history H\**. Neste caso, com a ocorrência de  $e$ , a entrada em  $K$  incidirá no estado mais recentemente visitado de  $K$ , independente do nível em que se encontrar. Desse modo, o novo estado poderá ser  $A$ ,  $B$ ,  $C$  ou  $D$ , conforme o passado recente de ativação.

## 2.7 Poder de expressão

Está além deste trabalho empregar várias linguagens para especificação de diálogo e comparar os resultados obtidos. Em [Har87] é feita uma extensiva comparação dos Estadogramas com outras notações de mesmo propósito. As principais diferenças enfatizadas são: a natureza gráfica (muitas notações são na forma de texto), mais intuitiva e facilita a compreensão; a deficiência de algumas notações no suporte à hierarquia e concorrência, e a dificuldade de manutenção das especificações.

A literatura envolvida com a especificação de diálogo é unânime em citar extensões dos diagramas de transição de estados. Entre as extensões têm-se ATN (*Augmented Transition Network*), RTN (*Recursive Transition Network*), Rapid/USE[Was85] e os diagramas descritos em [Jac86], por exemplo. Estas extensões enquadram-se no modelo de interação de redes de transição, conforme Green[Gre86]. Outros dois modelos citados são: gramáticas e eventos. Nenhum modelo está imune a críticas (pág. 60).

Green afirma que o modelo de eventos é o mais poderoso. Permite, por exemplo, a facilidade de descrever diálogos *multithread* (§5.4.2). Estadogramas podem representar diálogos *multithread*. Isto é possível devido à capacidade de descrever estados concorrentes e permitir a comunicação entre eles.

Uma dificuldade para Estadogramas é permitir algo equivalente à criação e destruição de um tratador de eventos (*event handler*, veja [Gre86] para detalhes). Com esta possibilidade poderíamos ter as operações de *cut* e *paste*, por exemplo, realizada entre instâncias de tratadores de eventos que são responsáveis por edição de textos. Tratadores de eventos ainda permitem o uso de variáveis, atribuições e sentenças com condições. Eles assemelham-se ao código de uma linguagem de programação convencional, e exigem necessariamente um programador para a descrição. Estadogramas são mais intuitivos.

Estas e outras questões, contudo, precisam ser devidamente consideradas em outros trabalhos. Observa outras notações segundo suas qualidades apenas para responder “O que está faltando nos Estadogramas?” Não tem o intuito de comparar notações quanto ao que podem ou não descrever. A subseção abaixo, contudo, mostram o poder de expressão dos Estadogramas comparados com os diagramas de transição de estados.

### 2.7.1 Estadogramas, uma extensão dos diagramas de estados

No capítulo 3 são comentadas várias notações para especificação de interfaces. Estadogramas são derivados dos diagramas de transição de estados, contudo, fornecem extensão que eliminam alguns inconvenientes destes. Abaixo segue uma lista contendo algumas dificuldades de expressão com os diagramas convencionais seguida de um exemplo que envolve todos os itens considerados.

Dificuldades com os diagramas de estados convencionais:

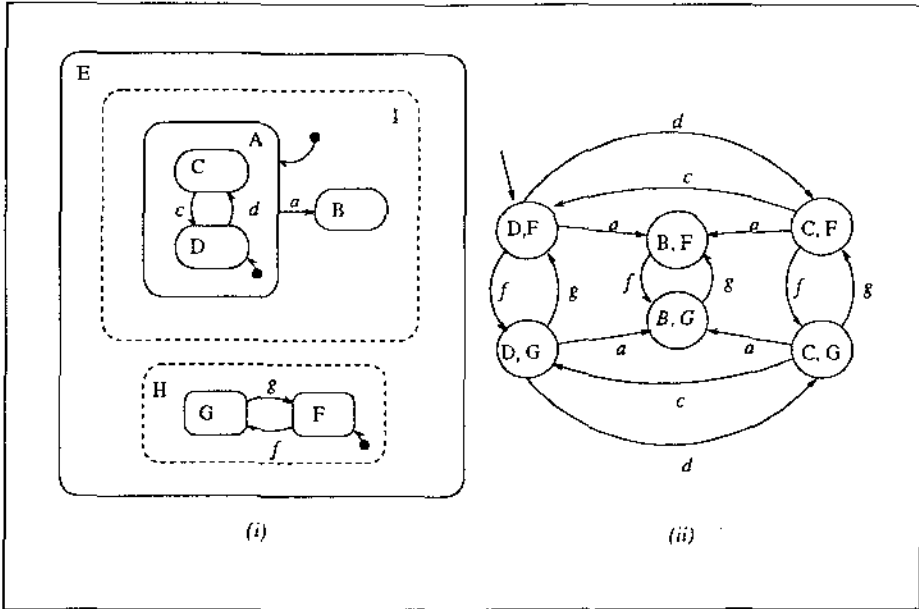


Figura 2.12: Estadogramas X diagramas de estados convencionais

1. Não fornecem noção de hierarquia, ou modularidade, em conseqüência não suportam o desenvolvimento *top-down* ou *bottom-up*.
2. Um evento que provoca a mesma transição de um grande número de estados deve ser explicitamente especificada para cada estado. Não há como “fatorar” transições semelhantes (ocasionadas por um mesmo evento) de um grupo de estados.
3. Representações de estados associadas a situações de um sistema em instantes de tempo são inviáveis. O número de estados cresce de forma exponencial, enquanto o sistema linearmente, pois cada possível estado deve ser explicitamente representado.
4. DTEs são intrinsecamente seqüenciais. Não fornecem uma representação natural<sup>8</sup> para concorrência.

A figura 2.12 mostra um Estadograma (i) e seu equivalente em diagramas de transição de estados convencionais (ii). Abaixo segue comentário relacionado a cada um dos itens acima e a figura 2.12.

Usando Estadogramas pode-se identificar a necessidade do estado A e especificá-lo posteriormente (*top-down*). Pode-se fazer o inverso, ou seja, identificar seus subestados C e D e encapsulá-los no estado A (*bottom-up*). Nota-se que para o estado B não interessa o que ocorre em A (modularidade), e a hierarquia existente desde o estado E até o estado C e D.

<sup>8</sup>Modelar um sistema concorrente pelo seu estado global não é considerado natural, neste trabalho.

Nota-se ainda que para os diagramas convencionais (ii) todas estas informações não estão explícitas!

A transição rotulada com o evento  $a$  (i) substitui quatro transições em (ii) induzidas pela hierarquia e concorrência existente nos Estadogramas.

Durante a especificação de um sistema pode ser identificada a presença adicional de um modo (pág. 84), não previsto anteriormente. Usando Estadogramas bastaria adicionar um estado concorrente ao conjunto de estados já especificados e tem-se o problema solucionado. Usando diagramas convencionais é preciso duplicar toda a especificação! Esta exemplo poderia ser equivalente à figura 2.12 antes que o estado  $H$  fosse incluído. É simples imaginar a especificação em Estadogramas antes do acréscimo do estado  $H$ , o que já não é tão simples com os diagramas de estados convencionais.

A situação anterior é conseqüência da dificuldade de se expressar concorrência nos diagramas convencionais. Apesar de estados como  $H$  e  $A$  serem independentes, os diagramas convencionais são incapazes de representar esta independência.

Neste exemplo ainda não foi explorado o mecanismo de comunicação entre estados concorrentes e transições condicionais. Acha-se suficiente, entretanto, para ressaltar a superioridade da clareza e concisão na notação dos Estadogramas.

## 2.8 Implementação

A implementação aqui é vista como a realização de uma especificação. É a implementação que torna concreto os benefícios de uma especificação em Estadogramas. Sem ela apenas podemos descrever um comportamento desejado, mas que não pode ser entendido por computadores. Por se tratar de um formalismo, os Estadogramas são possíveis de serem convertidos automaticamente em código executável. Esta seção trata desta conversão e como efetivamente os benefícios dos Estadogramas tornam-se reais na implementação de um núcleo reativo.

Ações podem controlar atividades ou gerar eventos internos afetando outros estados ortogonais. Naturalmente uma rotina pode consumir horas de processamento, entretanto, conceitualmente é instantânea. Este aspecto conceitual existe para preservar uma reação em cadeia. Como a implementação corrente é essencialmente seqüencial este problema não existe. A hipótese síncrona discorre mais sobre este aspecto.

### 2.8.1 Hipótese síncrona

A hipótese síncrona foi formulada em [BG92]. Ela assume que o sistema é infinitamente mais rápido que o ambiente, i.e., as respostas aos estímulos externos é sempre gerada no mesmo passo em que o estímulo é introduzido[PS91].



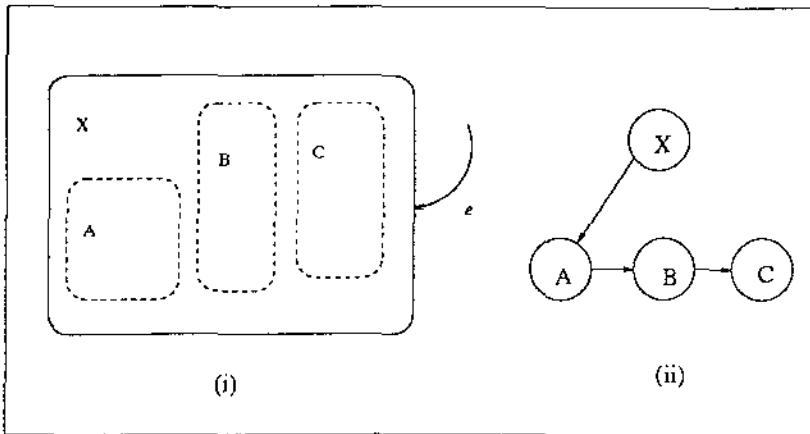


Figura 2.13: Exemplificando o determinismo.

Observe que uma longa seqüência de comunicação interna pode ser gerada como resposta. A seqüência, conforme esta hipótese, não será interrompida, i.e., nenhum novo evento será tratado enquanto o tratamento do evento corrente não for concluído.

Embora conceitualmente uma ação consuma zero unidades de tempo ela pode equivaler a uma rotina ou procedimento de uma linguagem de programação comum (como visto) e, portanto, consome tempo. Em [BG92] são feitos comentários acerca do quão real é esta hipótese. No nosso caso interessa garantir que uma reação em cadeia não será interrompida.

### 2.8.2 Comportamento determinístico

A implementação corrente é determinística. Devido à existência de várias situações de “conflito,” é preciso estabelecer claramente a semântica das especificações do ponto de vista operacional. Os benefícios do determinismo dificilmente podem coexistir com concorrência[BG92]. Esta seção, portanto, provê uma ordem de execução nas atividades concorrentes, o que pode ser aproveitada pelo indivíduo que constrói as especificações. Se este interesse não existir, o determinismo pode ser visto como um caso particular do não determinismo.

A figura 2.13 mostra um estado concorrente (X). Ele possui três subestados ortogonais: A, B e C. Em uma implementação não determinística a entrada em X pelo evento *e* poderia ser de várias formas. Se há concorrência poder-se-ia entrar simultaneamente nos seus subestados ou subconjunto dos mesmos. Assumindo uma implementação seqüencial pode-se ter uma permutação dos estados. Note que há a necessidade de um sorteio para estabelecer esta ordem. Para monoprocessadores esta abordagem é muito flexível e degrada desnecessariamente o desempenho. Não há como, exceto para casos particulares, reduzir o tempo de execução explorando o paralelismo potencial em máquinas com apenas um processador.

Determinismo, portanto, também implica em código mais eficiente. Para compreendermos melhor o que acontece em casos como o da figura 2.13, observe-se a árvore da figura 2.13

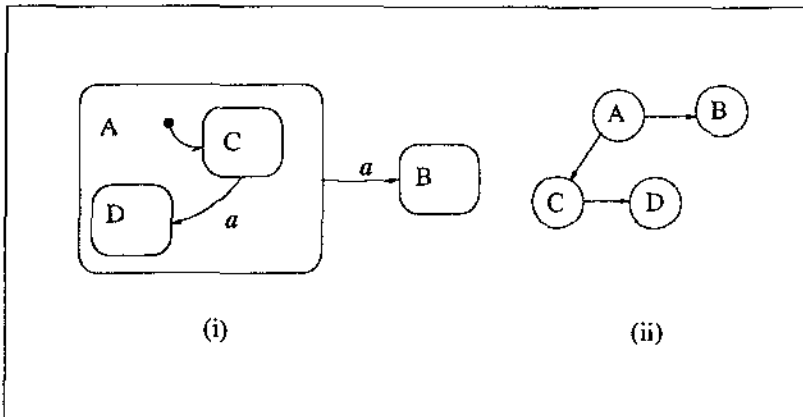


Figura 2.14: Outro exemplo de determinismo.

(ii). Esta árvore representa a hierarquia existente do estado  $X$  e é utilizada para definir o comportamento determinístico. Por exemplo, conforme desenhado, o “primeiro” filho de  $X$ , ou seja  $A$ , é o estado que tem precedência sobre os demais. Os demais referem-se aos seus irmãos.

De forma análoga,  $B$  tem precedência sobre todos os demais irmãos (apenas  $C$  neste caso). Voltando em (i) o evento  $e$  irá provocar, conforme as precedências estabelecidas pela árvore, a entrada no estado  $X$ , seguida das entradas em  $A$ ,  $B$  e  $C$ , nesta ordem. Para a saída também é utilizada a árvore, só que em sentido contrário. Ao invés de “descer” a árvore, caminha-se até a raiz, logo o estado  $X$  será o último estado a ser desativado.

Existem situações de “conflito” entre estados que não são concorrentes. Por exemplo, suponha que  $C$  (figura 2.14) seja o estado corrente; se o evento  $a$  ocorre qual o próximo estado ativo,  $D$  ou  $B$ ? A implementação corrente sempre percorre a árvore no sentido da raiz para as folhas. Quando é identificada alguma transição neste caminho ela é executada, sobrepondo-se a qualquer transição de seus subestados. Na figura 2.14, o primeiro nó a ser atingido é o  $A$  e, portanto, a transição será de  $A$  para  $B$ . Note que a transição de  $C$  para  $D$  jamais ocorrerá.

## 2.9 Ferramentas

Nesta seção são vistos alguns ambientes que apoiam a construção de sistemas reativos. Estes ambientes exemplificam os esforços e as atenções despendidos com os Estadogramas, além de ressaltar sua importância apesar do seu surgimento “recente.” Tradutor Blob e o editor/simulador Egrest são duas ferramentas usadas no presente trabalho.

Todos os sistemas abaixo citados possuem em comum o emprego de Estadogramas como linguagem central. Statemaster é um UIMS (§3.6) que usa Estadograma para especificação

do diálogo. *Statemate* é uma tentativa de se obter uma metodologia e ferramentas de suporte para o desenvolvimento de sistemas reativos onde *Estadogramas* é a principal linguagem. *Reacto* concentra-se na aquisição e implementação correta de especificações de software para sistemas reativos e também utiliza *Estadogramas*. Tradutor *Blob* permite a conversão de *Estadogramas* para código C e *Egrest* auxilia a edição e a depuração de *Estadogramas* (interage com o Tradutor *Blob*).

### 2.9.1 Statemaster

*Statemaster*[Wel89] é um UIMS (§3.6) utilizado na construção de protótipos e implementação de interfaces homem-computador. Fornece o *run-time* de uma interface correspondente ao controle de diálogo.<sup>9</sup>

O responsável pela especificação deve converter manualmente os *Estadogramas* gerados, também manualmente, para objetos compreendidos pelo *run-time* do *Statemaster*. O paradigma de objetos e C++ devem ser utilizados para esta finalidade.

A adoção de *Estadograma* como linguagem de especificação de controle fundamenta-se na complexidade dos diagramas de transição de estados usados comumente em outros UIMS's. Esta complexidade<sup>10</sup> torna a especificação intratável para interfaces sofisticadas.

### 2.9.2 Statemate

*Statemate* é um ambiente gráfico criado para a especificação, análise, projeto e documentação de sistemas reativos. O ambiente fornece três linguagens visuais (gráficas) e ortogonais: *module-charts*, *activity-charts* e *Estadogramas*. Cada uma representa uma visão diferente de um sistema. As linguagens capturam, respectivamente, a estrutura, a funcionalidade e o comportamento reativo. As linguagens contam com um editor gráfico que facilita a construção de especificações. As especificações podem ser analisadas (detecção de *deadlock* e não determinismo, por exemplo). Permite ainda execução (interativa), simulação (*batch*, especificada através de uma linguagem própria) e geração automática de código em Ada.

Em [HLN<sup>+</sup>90] encontra-se uma visão geral de *Statemate*. Particularidades podem ser obtidas em [iLI87] e [iLI89b].

### 2.9.3 Reacto

*Reacto* é um sistema de suporte a aquisição e implementação correta de especificações de software para sistemas reativos[mGGW89]. Para a modelagem dos sistemas reativos *Reacto*

<sup>9</sup> Equivale ao comportamento de um sistema reativo. O diálogo da interface é o que se denomina comumente de sintaxe da interação.

<sup>10</sup> A origem desta complexidade encontra-se em algumas características inconvenientes (§2.7.1) dos diagramas de transição de estados.

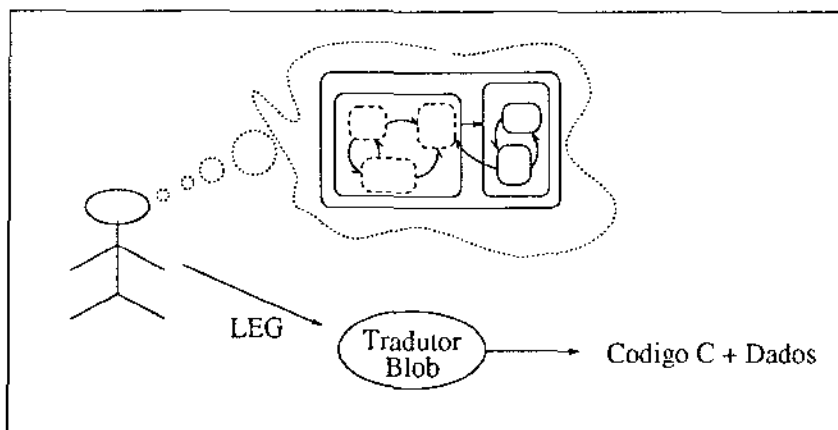


Figura 2.15: Ciclo simples de utilização do tradutor.

usa diagramas de transição de estados derivados de Estadogramas. Mantém algumas extensões dos Estadogramas, mas restringe, por exemplo, o mecanismo de *history*. Reacto está voltado para o projeto confiável de sistemas reativos. A confiabilidade advém de invariantes fornecidas pelo especificador associadas a estados, e através da prova de que a geração de código realizada pelo compilador é correta.

#### 2.9.4 Tradutor Blob e Egrest

Por todo este texto usa-se o termo Tradutor Blob<sup>11</sup> para designar um “compilador” e um *run-time*. O “compilador” gera código C<sup>12</sup> a partir de especificações em uma linguagem de alto nível e propósito específico. O *run-time* interage com o código gerado e interpreta os dados para realizar as atividades de um núcleo reativo. O Tradutor Blob e o *run-time* descritos em [Fil91b] foram adaptados até que mudanças propostas inviabilizaram novas adaptações, que não foram realizadas. Onde há a necessidade de distinguir um destes dois componentes usa-se apenas tradutor e *run-time*.

O Tradutor Blob[Fil91b] é um sistema voltado para a implementação de sistemas reativos. Compreende um *run-time* (parte invariante) e um tradutor. O tradutor converte descrições na linguagem LEG (descrição em forma de texto dos Estadogramas) em código C mais informações, ambos são utilizados pelo *run-time* para disparar as ações a medida que entradas são fornecidas. Portanto, a união do *run-time* com a saída do tradutor equivale a um núcleo reativo (§2.3). Trechos em código LEG são fornecidos neste trabalho e, inclusive, alterações são propostas. Esta dissertação não fornece uma descrição desta linguagem (LEG), pois é bem intuitiva e pode ser obtida em [Fil91b].

O sistema foi desenvolvido no Departamento de Ciência da Computação DCC - IMECC -

<sup>11</sup> *Blob* é a denominação dada em [Har87] para um estado descrito em Estadogramas.

<sup>12</sup> O *run-time* interage com este código e também está escrito em C.

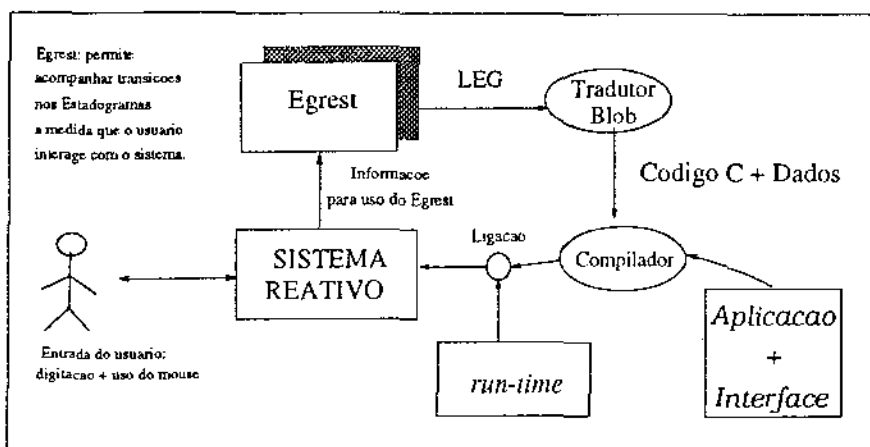


Figura 2.16: Relação entre Tradutor e Egest.

UNICAMP como parte de programa de mestrado. O código fonte do sistema está disponível, o que permite testar alguns acréscimos e modificações (§2.6). O protocolo de acesso ao núcleo reativo é muito simples compõem-se de poucas funções. As principais delas é `stEng(event)`, utilizada para passar para o núcleo reativo um evento lógico (`event`) obtido de um ou mais eventos físicos pela camada da interface (§2.3).

A figura 2.15 permite identificar o uso deste tradutor do modo mais simples. O responsável pela descrição do diálogo (boneco) produz manualmente a especificação em Estadograma e a converte, também manualmente, para uma linguagem equivalente denominada LEG.<sup>13</sup> O projetista então fornece código LEG para o Tradutor Blob que gera código C e dados que parcialmente equivalem à implementação dos Estadogramas. A saída do Tradutor (código C + dados) é então compilada e ligada ao *run-time* gerando o núcleo reativo. Todo o código resultante representa a implementação das especificações. A interface (§2.3) corresponde ao código responsável por gerar a entrada para os Estadogramas. A entrada é fornecida através de um protocolo que também fornece algumas funções úteis. A saída corresponde a chamadas de rotinas da aplicação.

A figura 2.16 mostra o processo anterior quando é feito o uso de outro sistema, o Egest (veja detalhes em [Eli92]). Neste caso o projetista constrói os Estadogramas e gera código LEG automaticamente através do Egest, eliminando a atividade manual e propensa a erros do caso anterior. O Egest ainda permite a simulação da especificação — “acompanhamento” realizado no *Egest* à medida que o sistema é executado. Desta forma o usuário pode interagir com o sistema reativo enquanto, simultaneamente, o Egest exibe o progresso do sistema através de transições entre estados.

Observe-se que o *run-time*, por ser invariante, pode ser colocado em uma biblioteca e somente ligado com o código objeto gerado pelo compilador. A aplicação e a interface

<sup>13</sup>Linguagem (na forma de texto) definida em [Fil91b] para descrição de Estadogramas. Serve como entrada para o Tradutor Blob. Este trabalho também propôs algumas mudanças nesta linguagem (capítulo 5).

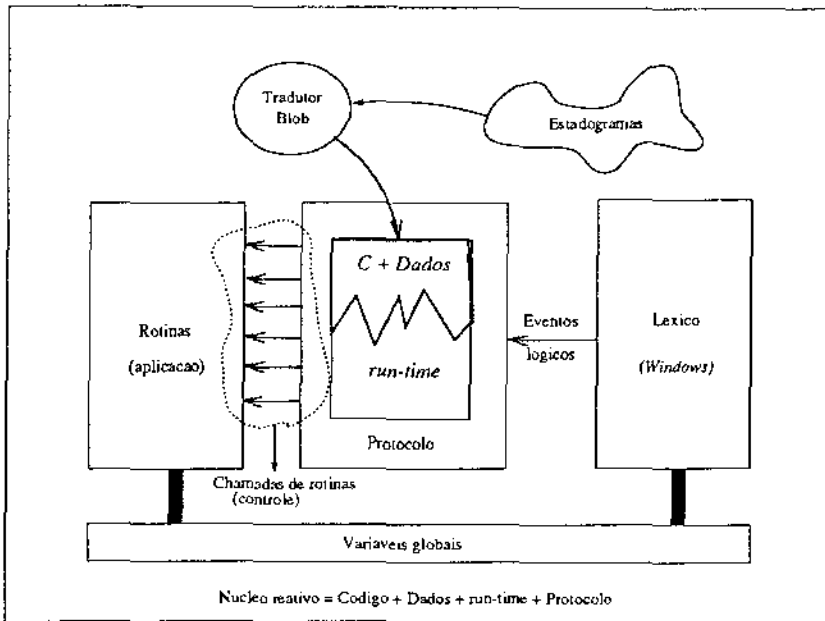


Figura 2.17: Arquitetura de código.

(quadrado no canto inferior direito) equivalem às camadas descritas em §2.3. O Egrest é usado apenas pelo projetista do software da interface (§3.5.7) durante o desenvolvimento. Enquanto interage com o sistema reativo ele pode observar o progresso da execução em termos de transições entre estados na especificação em Estadogramas. Não necessariamente o Egrest precisa ser executado na mesma máquina que o sistema reativo. Esta característica permite que seja observada a interação do usuário sem que ele saiba. Isto é útil para identificar dificuldades com a interação. A ausência física de um observador (até mesmo desconhecido pelo usuário) pode produzir resultados mais significativos que uma entrevista pessoal.

O Tradutor Blob e o Egrest são ferramentas importantes. Conforme Wasserman[WPSK86], diagramas de transição e leiautes de tela são inadequados para o entendimento do usuário na ausência de um interpretador destes diagramas.

### Processo de uso do Tradutor Blob

Na figura 2.17 vemos a arquitetura de software suportada pelo uso do Tradutor Blob e do *run-time* disponível.

A implementação corrente assume que o protocolo é utilizado seqüencialmente, sem concorrência. Isto vem ao encontro com a hipótese síncrona, pois apenas um evento por vez é fornecido ao *run-time* e o próximo só será fornecido após todas as ações serem executadas. O módulo léxico é o único que utiliza este protocolo.

Os módulos operam seqüencialmente e mantêm uma relação mestre/escravo da direita

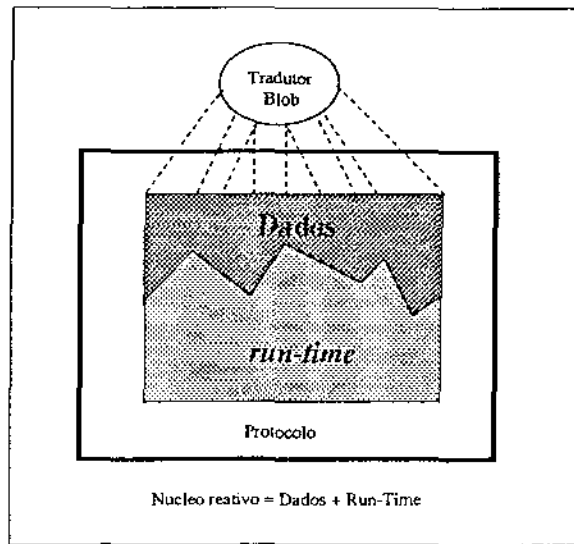


Figura 2.18: Nova arquitetura de código.

para a esquerda. Ou seja, as ações do usuário são manipuladas pelo módulo léxico (código em *Windows* que recebe e trata eventos de baixo nível). Ele gera eventos lógicos que são transferidos, via chamada de rotina (usando o protocolo), para o núcleo reativo. O núcleo reativo identifica quais as reações a serem tomadas em decorrência do evento recebido e da configuração corrente. Estas reações (ações) são chamadas à rotinas da aplicação (responsáveis pela semântica). Os resultados do processamento são colocados em variáveis globais (camada inferior) que podem ser obtidas pelo componente léxico. O componente léxico então reflete as alterações eventualmente ocorridas conforme o conteúdo desta memória.

Esta não é a situação ideal, pois o controle do sistema está exclusivamente nas mãos dos usuários. Em muitos casos o sistema deve ter um mínimo de controle.

A memória compartilhada é obrigatória para comunicação entre os módulos. Isto se deve à relação estrita de transferência de controle entre eles. O ideal seria permitir que o sistema também gerasse eventos lógicos para o núcleo reativo. Seria equivalente a fornecer a mesma relação estabelecida com o componente léxico.

Este comportamento impõe algumas restrições. A mais palpável é a impossibilidade de transferir o controle para a aplicação. Sistemas que utilizam estilos como o demonstracional<sup>14</sup> deve ter a possibilidade de em algum momento conduzir o diálogo. O uso de um *tutorial* serve de exemplo.

A figura 2.18 mostra uma evolução do processo anterior. A mudança mais importante é a inexistência de código gerado pelo Tradutor. Ele agora gera apenas dados.

<sup>14</sup>Estilo de interação recente que consiste numa extensão de outro estilo: manipulação direta. Veja §3.2 para mais detalhes.

Nesta nova situação o ciclo (mudanças, Tradutor Blob, compilador C, observação dos resultados) é evitado! O *run-time* precisa apenas de dados com esta mudança. Assim, em tempo de execução pode-se alterar a especificação e observar os resultados imediatamente. Isto estimula a experimentação e evita uma perda desnecessária de tempo. Seguramente facilita a síntese da especificação que pode ser construída interativamente, adicionando-se estados e transições entre eles.

Esta mudança, na verdade, apenas abriu caminho para que esta síntese alternativa bem como a alteração da especificação possam ser realizadas. Observe-se que muitas questões surgem quando se efetua mudanças na especificação do comportamento de maneira dinâmica. Por exemplo, seria permitido remover um estado corrente? Qual seria a semântica desta remoção?

Uma outra importante mudança foi a conversão do *run-time* para a linguagem C++. Nesta nova linguagem o paradigma de objetos é explorado e podemos ter a coexistência de vários estadoграмas em execução independente. Cada especificação é mantida por um objeto.



## 2.10 Resumo

Uma classe de sistemas é denominada de sistemas reativos por apresentarem um comportamento complexo. Este comportamento é difícil de ser descrito e Harel[Har87] propõe os Estadogramas para esta finalidade. As características principais da notação Estadogramas bem como alguns conceitos foram exemplificados. Isto inclui as alterações sugeridas neste trabalho e suas semânticas. Ferramentas que fornecem suporte ao uso de Estadogramas foram citadas. Elas mostram a necessidade e aplicabilidade desta notação.

Vimos como a arquitetura de um sistema reativo relaciona-se com a de uma interface (§2.4).

A atual implementação do *run-time* possui limitações:

- (1) Não prevê exceções, especialmente aquelas fruto de especificações inconsistentes.  
Não é função do *run-time* checar consistência de especificações, mas isto não justifica um comportamento imprevisto.
- (2) Uma análise das características não implementadas mostra que o código não está suficientemente estruturado para suportar novos recursos.

A linguagem LEG é uma forma alternativa de se registrar especificações em Estadogramas. Mudança nos diagramas implica em mudança na linguagem. Mudar a linguagem também implica em alterações no *run-time*, responsável pela execução da linguagem. Este capítulo descreveu algumas alterações propostas. No capítulo 5 também se encontram propostas de mudanças na linguagem LEG.

O capítulo não é exaustivo, diversas características de Estadogramas foram omitidas; mais exemplos e mais detalhes poderiam ser fornecidos. Entretanto, as referências citadas podem preencher estas lacunas e o essencial e fruto deste trabalho foi registrado.

O próximo capítulo discorre sobre interfaces homem-computador. A informação nele contida foi essencial para orientar o desenvolvimento da interface (parte deste trabalho) além de ser a base (fundamento) para grande parte das observações deste capítulo.

## Capítulo 3

# Interface Homem-Computador

*“Embora o projeto de interface em parte seja uma arte, pode-se pelo menos sugerir uma abordagem organizada para o processo.”*  
Foley et al. [FvDFH90, pág. 429]

O capítulo anterior discorreu sobre os Estadogramas, notação usada durante o desenvolvimento de uma interface. Este capítulo mostra a noção de interface aqui empregada. Há uma ênfase na perspectiva de um projetista de software. São descritos conceitos, métodos e técnicas atualmente em uso no projeto e implementação de interfaces. Não se trata de um texto introdutório sobre o assunto. Recomenda-se [BC91] e [HH89a] como leituras iniciais.

Este capítulo tem como objetivos: orientar o desenvolvimento de uma interface; identificar o que diretamente ou não afeta uma linguagem candidata a especificação de diálogo (Estadogramas) e, sobretudo, estabelecer um panorama da área onde Estadogramas são empregados.

Após a definição de alguns termos o texto discorre sobre estilos de interação; qualidades buscadas em um interface; princípios; técnicas; ciclo de vida; modelos de interação e arquitetura de software para interfaces; e questões relacionadas com a implementação de interfaces.

---

### 3.1 Terminologia básica

Esta seção define alguns termos. O primeiro é diálogo, geralmente usado com a mesma acepção de interface, também definido.

É difícil separar a interface (software) da aplicação. Muitos acham que não há distinção entre estes elementos, e outros consideram difícil até mesmo imaginar tal divisão [Mye89].

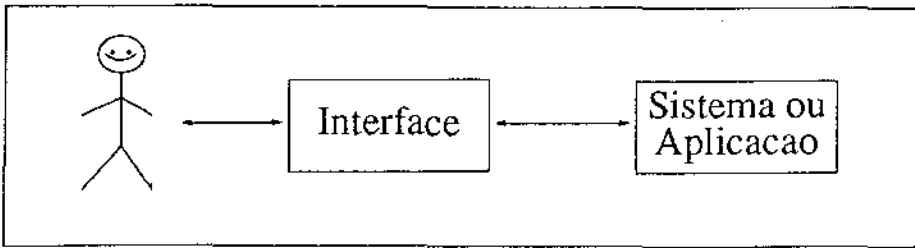


Figura 3.1: Visão simplificada de um sistema interativo.

---

**Diálogo:** comunicação existente entre o ser humano e o sistema de computação — é a troca de símbolos e ações entre o usuário e o computador.

---



---

**Interface:** meio (hardware e software) através do qual o diálogo é efetivado[HH89a].

---

Interessa a este trabalho o que diz respeito ao componente software de uma interface. A figura 3.1 permite identificar o módulo pertinente à interface em um sistema interativo. Sistema ou aplicação caracteriza o módulo responsável pela transformação de dados de entrada em saída, ou seja, o aspecto funcional (relativo a algoritmos) de sistemas interativos. Neste trabalho interface deve ser entendido como o componente (software) responsável por receber a entrada do usuário, realizar algum processamento sobre a mesma (por exemplo, mapear uma escolha de um item de menu para um comando reconhecido pela aplicação), e entregá-la para componentes que não fazem parte da interface, i.e., o sistema ou aplicação. No sentido contrário a interface deve ser sensível ao estado do sistema, refletir e permitir que se tenha acesso a toda e qualquer informação que seja de interesse do usuário.

O único meio do usuário ter acesso a funcionalidade ou semântica da aplicação é através da interface. O sistema ou aplicação preocupa-se essencialmente com as transformações sobre dados. Não é de seu interesse considerar como a entrada foi obtida do usuário, ou processamentos realizados para dar forma ao resultado a ser exibido para o usuário. Definir o que é exatamente função de um ou outro módulo não é tarefa simples.

---

---

**Modelo de imagem:** modelo utilizado para expressar a saída. Este modelo estabelece como imagens são descritas. Um dos modelos mais comuns é o de *pixel* (*PIcture X ELeмент*) — a tela é vista como uma coleção de pontos cuja cor pode ser alterada.

---

---

---

---

**Sistema de Janela (*Window System*):** caracteriza-se por ser responsável pelas seguintes funções: definir um terminal abstrato que fornece independência de dispositivo e permite compartilhá-los; fornecer um modelo de imagem para expressar a interação com o terminal abstrato, que gerencia recursos associados a dispositivos de entrada e saída.

---

---

Seguramente uma das características mais visíveis de um sistema de janela é o serviço que fornece para permitir que um único dispositivo físico possa ser compartilhado entre “clientes” deste dispositivo. Por exemplo, em um único terminal físico pode-se ter disponível várias ocorrências de terminais abstratos, cada um associado a uma janela. Isto é o que ocorre quando tem-se uma tela dividida em várias janelas. Cada uma destas janelas corresponde a um terminal abstrato. Se tem-se vários terminais abstratos usando a mesma tela, mouse, teclado, e outros, é preciso que o sistema de janela, por exemplo, identifique qual terminal abstrato está recebendo a entrada. Em [SG86] é comentado um sistema muito utilizado: o *X Window*.

---

---

**Gerenciador de janela [Mye88]:** componente de um sistema de janela. Trata-se de software que permite o usuário gerenciar diferentes contextos através da separação física de áreas de uma ou mais telas. Por exemplo, em ambientes multiprogramados é comum o operador trabalhar em mais de uma atividade simultaneamente, i.e., disparar a execução de várias aplicações, nem por isso, precisa-se ter tantos terminais quanto for o número de aplicações. Este gerenciador é responsável pela coexistência de várias janelas em um mesmo terminal onde cada uma pode ser vista como um terminal virtual.

---

---

É comum o uso do termo gerenciador de janela para caracterizar a interface do usuário e sistema de janela para a interface de programação [MR92].

---

---

**GUI[KA90]:** acrônimo de *Graphic User Interfaces*, refere-se a sistemas que permitem a criação e manipulação de interfaces através de eventos de janelas, menus, ícones, caixas de diálogo (*dialog boxes*), mouse e teclado. Neste sentido GUI é equivalente a um sistema de janela.

---

---

Microsoft *Windows* e *X Window* são exemplos de GUIs. Geralmente as GUIs têm como base ambientes multiprogramados e permitem associar cada janela (*window*) a um processo.

## 3.2 Estilos de Interação

De maneira geral pode-se identificar algumas formas através das quais os seres humanos interagem com os computadores. Esta “forma” equivale ao vocábulo *feel* da expressão *look and feel*. É a linguagem usada pelo usuário para se comunicar com o computador. *Look* refere-se a apresentação. A apresentação é tudo que um observador externo pode perceber como reação do computador (§4.3). Os estilos abaixo apresentados podem ser obtidos em detalhes em [FvDFH90, Shn87]. A figura 4.6 na página 74 exemplifica uma implementação de vários estilos em uma mesma interface.

Estilos estão intrinsecamente ligados ao comportamento da interface e apresentam dificuldades para a sua especificação. Alguns são simples de serem descritos, por exemplo, uma linguagem de comandos pode ser facilmente descrita usando-se BNF. Manipulação direta é um dos mais sofisticados e muito trabalho tem sido realizado para especificá-la adequadamente [Jac86, FBS7, Hil87].

O projetista da interface (§3.5.7) é o indivíduo responsável por identificar qual o estilo adequado para cada caso ou se uma combinação deles é necessária. A decisão depende de vários fatores e está além dos objetivos deste trabalho. Geralmente, contudo, segue-se alguns princípios de projeto (§3.5.2).

**WYSIWYG** Uma interface WYSIWYG (*What You See Is What You Get*) permite que o usuário veja através da tela o resultado do processamento, ou seja, a saída. Por exemplo, um editor de texto WYSIWYG mostra na tela o que será impresso, palavras em negrito aparecem em negrito, vários tamanhos e tipos de letras são distintos, proporções são preservadas e assim por diante. Isto possibilita o usuário observar o resultado de uma impressão antes de obtê-la realmente. Contraste com outros editores (não WYSIWYG) onde basicamente o usuário observa uma seqüência de caracteres recheados de caracteres de controle.

**Linguagem de comandos.** Meio de interação tradicional em que o usuário fornece via teclado uma seqüência de caracteres correspondentes a entrada.

**Linguagem natural.** Extensão do caso anterior onde o usuário não está limitado a um vocabulário exíguo de palavras e sintaxe rígida. Esta técnica torna qualquer pessoa um usuário de computador em potencial, pois a interação dá-se pelo diálogo do dia-a-dia. Em verdade existem algumas limitações e inconvenientes. São aplicáveis apenas a áreas do conhecimento bem específicas.

**Manipulação direta.** Conforme Shneiderman[Shn83] este termo está associado a algumas idéias centrais: (1) visibilidade do objeto de interesse; (2) ações rápidas e reversíveis; (3) visualização imediata do resultado, e (4) substituição de estilos de interação como linguagem de comandos e menus pela manipulação direta do objeto de interesse. Em outras palavras, permite que o usuário manipule, geralmente com o uso do mouse, representações da realidade. Por exemplo, sistemas de janela permitem que o usuário desloque uma janela, diminua ou aumente o seu tamanho com o uso do mouse, sem a necessidade de lembrar de um comando ou escolher uma opção em um menu. Este estilo representa um desafio à implementação e é considerado atentamente em §5.4.

**Demonstracional.** Do inglês *demonstrational*. Estilo no qual o usuário pode usar exemplos, fornecidos através de manipulação direta, para especificar operações abstratas. Para isto o sistema usa inferências para “sugerir” generalizações. Por exemplo, uma interface demonstracional pode notar que o usuário novamente “arrastou” um arquivo com extensão .bak até uma lata de lixo e, imediatamente, confirmar com o usuário se é desejável remover todos os arquivos com esta extensão. Da mesma forma que os demais estilos, não se aplica a todos os casos. Myers introduz o termo e o relaciona com outros em [Mye92].

**Ícônico.** Ícone é um símbolo gráfico que apresenta uma relação de semelhança ou analogia com um objeto, propriedade ou ação. Em interfaces icônicas conceitos são ligados a ícones. Geralmente o usuário tem acesso a estes conceitos simplesmente selecionando-os com o mouse, por exemplo.

**Menus.** São empregados para facilitar o diálogo para usuários inexperientes. Menu é uma lista de opções dispostas em uma coluna. Reduz a necessidade de memorização e limita o conjunto de opções. Ainda permite que só sejam selecionadas opções válidas em determinada situação. Hopkins[Hop91] apresenta os menus *pie* (menus circulares) que permitem a mesma facilidade de acesso a todos os itens, o que não é possível com os menus tradicionais. Os últimos geralmente trazem opções mais comumente selecionadas no início para reduzir o tempo de escolha.

**Form fill-in.** Também conhecido por *fill-in-the-blanks*. Seu emprego é adequado para a entrada de dados composta por vários campos. O usuário pode alternar entre os campos e fornecer a entrada em qualquer ordem. Os campos são identificados por rótulos. Vantagem: elimina a necessidade de lembrar de todos os argumentos de um comando, por exemplo.

### 3.3 Modelos

*“A inexistência de uma estrutura para as partes componentes de uma interação homem-computador conduz a procedimentos de desenvolvimento que são ad hoc e não-estruturados”*  
Hartson e Hix[HH89a]

No domínio de interfaces existem vários modelos.<sup>1</sup> Em [Nor91] há uma consideração mais exaustiva sobre modelos. Abaixo segue uma descrição concisa de alguns. Modelos são utilizados para orientar a especificação de algumas atividades de projeto (veja as fases de Definição de requisitos e Especificações do ciclo de vida de uma interface, §3.5.7).

*Modelo Cognitivo* é o modelo de como a mente do usuário trabalha. É obtido por especialista em cognição. Seu objetivo é compreender os limites e capacidades do usuário. Esse conhecimento geralmente é traduzido em princípios de projeto.<sup>2</sup>

*Modelo Conceitual, Modelo Conceitual do Usuário*, ou ainda *Modelo do Usuário* é uma representação do sistema formulada pelo projetista e fornecida ao usuário para facilitar o entendimento e uso do sistema — define objetos, relações entre objetos e operações sobre eles. Serve também ao projetista da interface durante o projeto da interface. Este modelo auxilia o aprendizado e desempenho do usuário[Nor91, pág. 228]. O modelo de protocolo virtual (descrito abaixo) embora tenha surgido com outra finalidade serve como modelo conceitual.

*Modelo de Interação*[Nie86] ou *Modelo Estrutural*[HH89a] consiste da descrição do processo geral de comunicação entre homem e computador, i.e., descreve de forma genérica e teórica a estrutura da comunicação do usuário com o computador. Esse modelo pode guiar o projetista durante o processo de representação do diálogo. Como exemplos de modelos estruturais pode-se citar os modelos lingüísticos: o modelo de Foley/van Dam[FvD82] (voltado para o projeto *top-down* de uma interface) e o Modelo de Protocolo Virtual[Nie86]. Em [Nie86] encontra-se uma comparação entre outros modelos e o modelo de protocolo virtual. Abaixo segue uma sucinta descrição do modelo de protocolo virtual e o Modelo de Seeheim. O primeiro não reflete a implementação, apenas serve de estrutura para discussão de aspectos da interface. O modelo de Seeheim, muito comentado na literatura, exhibe uma arquitetura de software para interface.

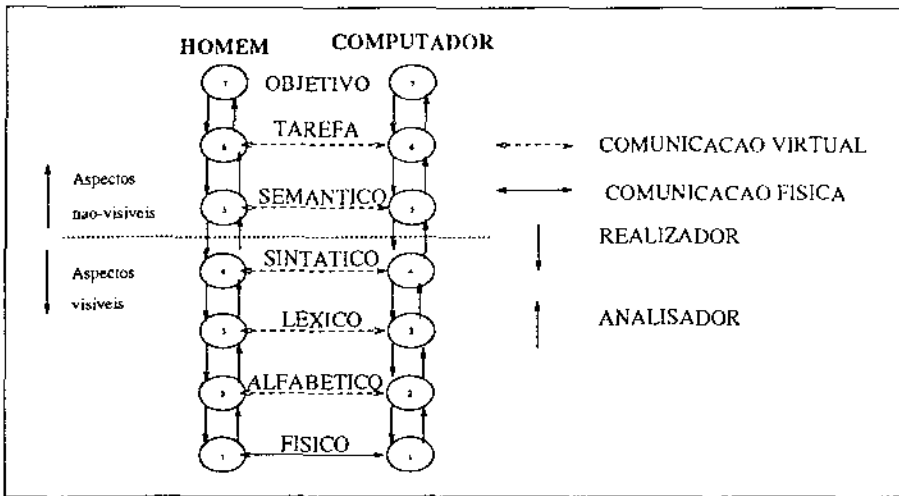


Figura 3.2: Modelo de Protocolo Virtual.

### Modelo de Protocolo Virtual (MPV)

No MPV[Nie86] (fig. 3.2) a interação homem-computador é vista como um diálogo no qual dois participantes trocam mensagens entre si seguindo convenções preestabelecidas. Nielsen[Nie86] afirma que discussões de fatores humanos da interação homem-computador tendem a ser imprecisas e centralizam-se em questões menores. Ainda acrescenta que isto é devido à carência de uma estrutura (*framework*) comumente aceita para tais discussões — falta uma taxonomia para a interação homem-computador. Talvez o mais importante, conforme Nielsen, seja um modelo para esta interação. Nielsen então propõe um modelo de interação entre o ser humano e um computador — o principal objetivo é fornecer uma estrutura sobre a qual discussões acerca de fatores humanos possam ser realizadas. Embora fatores humanos não seja a principal preocupação neste texto, este modelo é descrito sucintamente e foi útil durante o projeto aqui desenvolvido. Outros modelos citados em [Nie86], não apresentam a visão tanto do usuário quanto do computador, como feito pelo MPV. Esta característica adicional facilita a escolha deste modelo.

O modelo é inspirado no modelo de referência OSI/ISO usado em redes de computadores. Compreende uma hierarquia de diálogos virtuais, pois a comunicação física só ocorre no nível 1. Apresenta o lado do usuário e o do computador.

<sup>1</sup>Um modelo é uma representação simplificada de um sistema e serve como meio conciso de comunicação. Tende a ser abstrato e geral. É apenas uma representação e não o próprio sistema. Sem os modelos ter-se-ia que conduzir experiências reais sobre sistemas acabados e não se teria o benefício de avaliar alternativas de forma barata e rápida. Veja *Computer Simulation and Modeling* de Francis Neelambkavil. John Wiley & Sons Ltda. — 1987.

<sup>2</sup>Por exemplo, o número mágico  $7 \pm 2$  é um princípio utilizado para evitar a sobrecarga do usuário. Esta faixa de 5 a 9 é considerada normal, por especialistas pertinentes, para o número de atividades simultâneas que um ser humano consegue conduzir.



Todo sistema contém um alfabeto de unidades primitivas que carregam informação, mas não possuem significado isoladamente. Tudo que caracteriza tais unidades é detalhado na camada alfabética (letras, dígitos, linhas, cores, e outras). Os símbolos (*tokens*) trocados no nível léxico podem ser palavras, símbolos especiais, ícones, números, e assim por diante — são as menores unidades que possuem significado próprio. Tais *tokens* são compostos por elementos da camada alfabética. A camada sintática é responsável por estabelecer a seqüência de símbolos de entrada e saída trocados no nível inferior tanto no tempo quanto no espaço, isto inclui, p.ex., leiaute da tela. A camada semântica determina a funcionalidade detalhada do sistema, i.e., o que cada operação faz com cada objeto. O nível de tarefas estipula os tipos de objetos e operações existentes no sistema. Não necessariamente estão disponíveis diretamente, mas podem ser obtidas através de seqüências de operações do nível 5. Por último, a camada de objetivos distingue-se das demais por trabalhar com conceitos do mundo real.

### Modelo de Seeheim

O Modelo de Seeheim (figura 3.3) é um modelo abstrato de um UIMS. O modelo é lógico sendo, contudo, a função desempenhada por cada um dos módulos imprescindível em um UIMS[Gre85]. Em outras palavras, as funções desempenhadas pelos módulos devem ser previstas na construção de uma interface. É um modelo de arquitetura de software para interfaces. Componentes lógicos de um sistema interativo são delineados na figura 3.4. Este modelo nada mais é do que uma proposta de organização destes componentes em uma arquitetura de software — um modelo de estrutura de software em tempo de execução (*run-time*).

O modelo compreende o componente de apresentação que contém detalhes dependentes e específicos do dispositivo de saída, bem como a descrição do estilo de interação (nível léxico). O componente de controle de diálogo é responsável pelo processamento do diálogo e seqüência das operações (nível sintático), enquanto o modelo de interface da aplicação liga a interface à aplicação (nível semântico) através de chamadas de procedimentos e estruturas de dados. Embora existam críticas a este modelo[SV91, Mye89], ele continua sendo utilizado[SG91].

A caixa sem nome (figura 3.3) é um reconhecimento de casos onde é desnecessário o tratamento de informações pelo módulo de controle do diálogo. Nestes casos o módulo de controle de diálogo é evitado por considerações de eficiência. Existem outros problemas com o modelo de Seeheim[SV91]. Independente da complexidade de uma interface existem apenas três componentes. Isto pode naturalmente conduzir a elementos complexos e volumosos. Outro problema reside na retroalimentação semântica — o uso de manipulação direta provoca uma comunicação intensa entre estes elementos, o que nem sempre é viável. As referências citadas apresentam mais detalhes das dificuldades com este modelo.

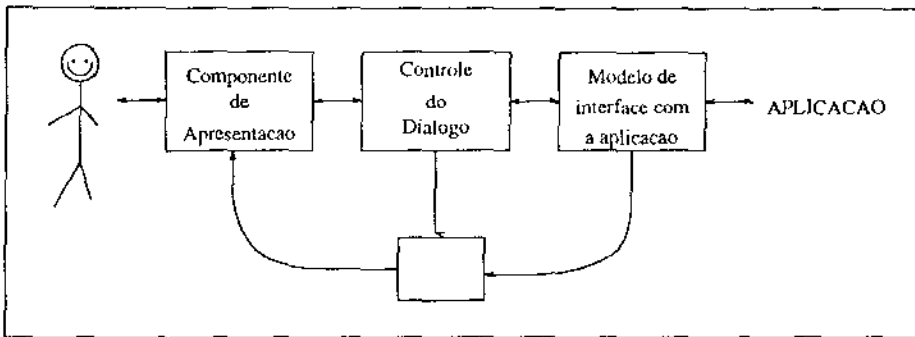


Figura 3.3: Modelo de Seeheim.

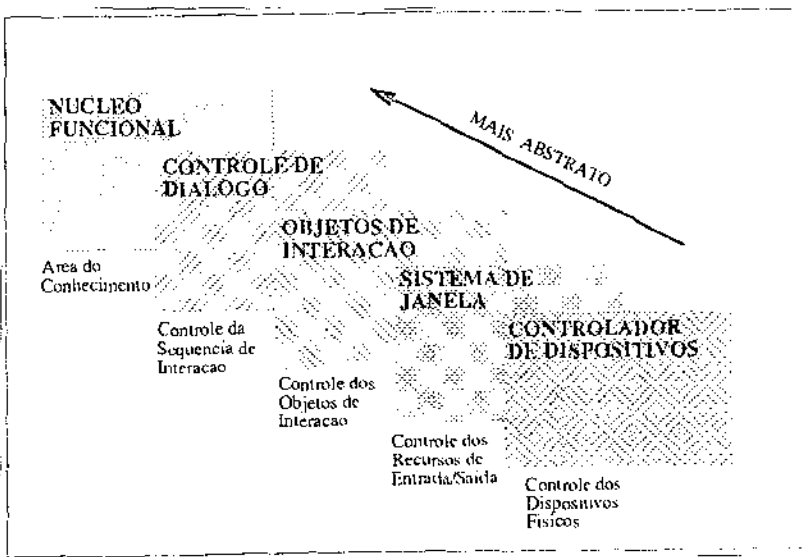


Figura 3.4: Arquitetura de software para interfaces.

### 3.4 Uma taxonomia para o software

Para o desenvolvimento de uma interface (§4.3), um sistema interativo pode ser visto segundo a arquitetura de software da figura 3.4. Nesta figura um sistema interativo é composto por várias camadas. Abaixo segue uma taxonomia para estas camadas obtida de [BC91]. Esta divisão em camadas é relacionada a outros aspectos na seção 2.3, figura 2.3.

Níveis:

1. *Controlador de dispositivos.* Um *controlador de dispositivos*, por exemplo, envia instruções a nível de pixel para exibir o cursor na posição desejada.
2. *Sistema de janela.* Controla e gerencia recursos físicos usados nas interações com o usuário. Fornece abstrações usadas para a interação. Sistemas de janela também são

vistos na página 43.

3. *Objeto de interação.* Sistemas de janelas fornecem serviços de nível relativamente baixo. Uma camada sobre estes serviços faz-se necessário para fornecer mais abstrações. Objeto de interação é a entidade que o usuário pode perceber e manipular com dispositivos de interação física tais como o mouse, ou teclado. Compreende entrada, saída e um comportamento que podem ser modificados através de atributos. Menu é um exemplo de objeto de interação.
4. *Controlador de diálogo.* Controla a seqüência de interação com o usuário e realiza os mapeamentos necessários entre o núcleo funcional e os objetos de interação.
5. *Núcleo funcional.* Implementa o conhecimento pertinente a determinado domínio. Esta camada é a que chama-se de sistema ou aplicação, ou seja, parte do sistema interativo que não tem relação direta com a interface com o usuário. Trata-se da parte do sistema independente dos meios utilizados para expressar a saída e é responsável pela sua funcionalidade.

### 3.5 Engenharia de software para interfaces

A engenharia de software estabelece algumas qualidades desejáveis a serem atingidas por um software (correção, confiabilidade, robustez, facilidade de manutenção e outras) e um conjunto de princípios (divisão de responsabilidades, modularidade, abstração e outros) que buscam, quando seguidos, atingir estes objetivos[GJM91].

Para aplicar princípios o engenheiro de software deve estar equipado com técnicas apropriadas. Técnicas podem ser agrupadas formando uma metodologia. O objetivo de uma metodologia é suportar uma abordagem para solução de um problema. Ferramentas, por sua vez, são desenvolvidas para suportar a aplicação de técnicas, métodos e metodologias. Ghezzi et al.[GJM91] fazem distinção entre métodos e técnicas. Métodos são regras gerais que governam a execução de alguma atividade; são abordagens sistemáticas e disciplinadas. Técnicas são atividades mais mecânicas.

Draper e Normam em [DN85] fornecem analogias entre o estado da arte de engenharia de software e o de projeto de interfaces. Este não é o objetivo desta seção. Entretanto, usa-se a taxonomia descrita acima (qualidades, princípios e assim por diante) para classificar informações relevantes ao desenvolvimento de interfaces. As informações são extraídas de várias fontes citadas e não representam uma simples transcrição. Informações adicionais e correlatas são fornecidas.

*A facilidade de usar e a de aprender* resumem os objetivos que usuários desejam encontrar em um sistema interativo. Pesquisas em interação homem-computador buscam satisfazer estes objetivos através de dois grupos principais — ciência da computação e psicologia[Abo91]. Por um lado tem-se a atenção focalizada em fatores humanos. Por outro, têm-se projetistas

de software e hardware dirigidos para aspectos de computação. No desenvolvimento realizado aspectos psicológicos são apenas utilizados conforme preconizados pela literatura.

Em [FvDFH90, págs. 391-414] qualidades e princípios podem ser vistos em mais detalhes.

### 3.5.1 Qualidades em uma interface

- *Facilidade de usar e aprender.* Embora seja auto-explicativo para os usuários, este termo precisa de um significado mais útil do que simplesmente fácil para projetistas. Habermann[Hab91] fornece uma semântica para os projetistas.
- *Taxa de erro.* Identifica o número de erros por iteração. Em alguns sistemas críticos (nucleares, controle de tráfego aéreo e outros) não deve ser fácil provocar erros por parte do usuário.
- *Recordação rápida (rapid recall).* O usuário deve lembrar-se rapidamente como usar o sistema. Ou seja, o modelo conceitual (§3.3) não deve apresentar resistência para um uso esporádico.
- *Atrativo.* Viu-se no capítulo 1 que nem sempre o sistema mais poderoso é o que será escolhido pelo usuário. É um termo ligado ao mercado, o usuário pode preferir um sistema mesmo com um desempenho inferior a outro.

Naturalmente os itens acima não se aplicam a todos os casos. Ao projetista cabe identificar as nuances e como serão tratadas. As qualidades e nuances devem ser identificados na fase de especificação de requisitos (§3.5.7).

### 3.5.2 Princípios para garantir a qualidade

Abaixo seguem princípios de projeto consensualmente empregados. Novamente tratam-se de elementos a serem estudados para cada caso particular. Embora Shneiderman[Shn87] faça referência a “regras de ouro” para denominar esta lista, nada assegura que estes princípios conduzirão à melhor interface[Mye89, MN90]. Russell et al.[RXW92] citam situações em que se tem conflitos entre eles.

- *Consistência.* Sistema em que o modelo conceitual (§3.3), funcionalidade, seqüência e o uso do hardware seguem regras simples e não apresentam casos especiais ou exceções. Ou seja, ações particulares do sistema devem sempre ser obtidas com ações particulares do usuário.
- *Retroalimentação (Feedback).* Ações do usuário devem gerar uma retroalimentação. Geralmente isto é obtido através de representações visuais que refletem, constantemente, o estado interno do sistema e, por conseguinte, suas alterações. Portanto, mantém o usuário constantemente informado do que está acontecendo.

- *Minimizar possibilidades de erro.* Deve ser fornecido ao usuário somente os comandos disponíveis no instante da interação. Assim, se uma operação não pode ser disparada o item correspondente do menu deve ser diferenciado, por exemplo.
- *Fornecer um meio de recuperação de erros.* Todos nós comete-se erros. Deve ser possível retornar ao estado anterior (*undo*), cancelar, interromper ou substituir um parâmetro ou comando. Sem estas operações a dificuldade de corrigir um erro pode tornar inaceitável um sistema, ou inibir a experimentação por parte dos usuários.
- *Tratar adequadamente usuários com habilidades diferentes.* Alguns sistemas são usados por uma grande variedade de pessoas, desde novatos até especialistas. De modo grosseiro alguns estilos de interação identificam-se com o grau de habilidade dos usuários. Por exemplo, menus são mais apropriados para os novatos enquanto os mais experientes obtêm melhor desempenho com linguagem de comandos. Uma interface flexível (pág. 54) fornece uma solução para a coexistência de vários estilos de diálogos.
- *Minimizar a necessidade de memorização.* Seguramente quanto mais memória for exigida para operação de um sistema mais se estará violando os objetivos vistos anteriormente. Quem não conhece um editor com “estranha” combinação de teclas? Menus e *form-fill* são estilos utilizados para esse propósito.
- *Metáforas.* O uso de metáfora para projeto de interface tem o intuito de diminuir a dificuldade com a interação. A idéia é utilizar ações, procedimentos e conceitos que se assemelham àqueles usados pelo usuário no ambiente real, p. ex., projetar um sistema de informação de escritório usando a metáfora de uma mesa de trabalho (*desktop*). A complexidade não é reduzida, mas ocorre um maior grau de familiarização do usuário com o sistema. Detalhes sobre metáforas podem ser obtidos em [CMK88].

Os princípios vistos são do âmbito de projeto de interfaces. Implementar estes princípios é outra questão.

### 3.5.3 Técnicas

Técnicas são recursos empregados para permitir a realização de determinado intento. Independência de diálogo é primordial para o projeto iterativo de interfaces, enquanto interface flexível diz como pode-se tratar adequadamente usuários com distintas habilidades. Abaixo seguem técnicas para auxiliar o desenvolvimento de interfaces.

#### Independência de Diálogo

O desenvolvimento de uma interface sem a existência de técnicas adequadas faziam da interface e do sistema módulos altamente entrelaçados, o que tornava difícil a alteração em qualquer um destes módulos[HH89a]. Isto conduzia a produtos de qualidade pobre, pois

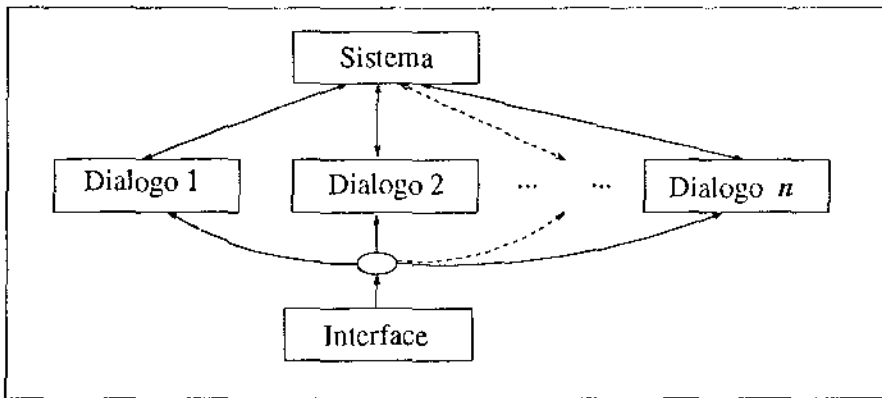


Figura 3.5: Independência de diálogo

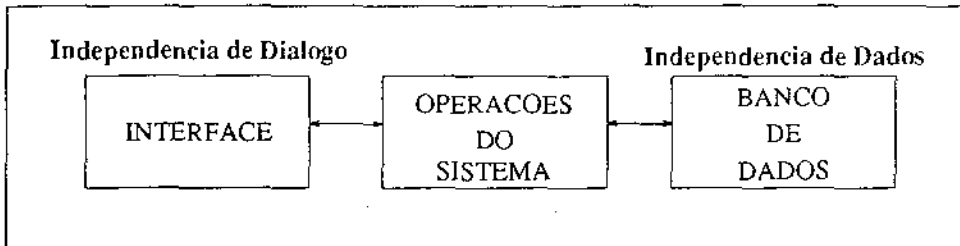


Figura 3.6: Independências de diálogo e dados em um sistema[WPSK86].

desenvolver uma interface é um processo cíclico de projeto e avaliação (§3.5.7). Projetistas de bases de dados enfrentam um problema similar quando se deseja modificar facilmente os dados sem alterar os programas. A solução encontrada foi a independência de dados. Surge então, um conceito análogo, **Independência de Diálogo**.

---

**Independência de diálogo:** separação entre a interface e a aplicação através de um protocolo de comunicação entre os mesmos.

---

Neste caso, a mudança em um destes elementos não afeta o outro contanto que o protocolo de comunicação não seja afetado. Existem vários modelos para software de interface que exibem esta independência como em §3.4. Modelos como o de protocolo virtual e o modelo de Seeheim também seguem esta independência (§3.3). Em [HH89a] é comentada exaustivamente esta independência. A figura 3.6 mostra a estrutura de um sistema interativo que apresenta a independência de dados e a de diálogo.

Esta flexibilidade tem muitas vantagens, por exemplo, pode-se criar interfaces diferentes para serem usadas por pessoas com conhecimentos diferentes, com variados níveis de habilidade (seja no uso do computador ou domínio do conhecimento da área da aplicação, i.e.,

especialistas ou novatos), de línguas diferentes, e assim por diante. A figura 3.5 exemplifica esta situação. Nela vê-se vários tipos de usuários, cada tipo servido por uma interface distinta. Independência de diálogo permite que um sistema possa ser usado por comunidades diferentes de operadores com interfaces distintas para cada uma delas mantendo, entretanto, um núcleo funcional comum. Ainda permite que estas partes sejam desenvolvidas isoladamente, onde cada especialista pode trabalhar na área que lhe é peculiar (veja [TB85]).

Tornar real estes benefícios exige algumas considerações e inconvenientes. Conforme [Har89]: “Quase todos concordam com a separação entre o componente de diálogo e o de computação. A separação ideal, contudo, é difícil de definir e mais difícil ainda de ser obtida.”

---

---

**Como obter:** separar em tempo de projeto o diálogo e o software de computação (aplicação).

---

---

### Interface flexível

---

---

**Interface flexível<sup>3</sup>:** denominação dada às interfaces que permitem a coexistência e cooperação entre vários estilos de interação (§3.2) numa mesma interface.

---

---

Uma interface pode ser vista como uma linguagem. Uma interface flexível seria aquela que apresentasse várias estilos (cada qual com sua linguagem correspondente) de interação para uso simultâneo e possivelmente intercalado. Para isto é necessário a existência de um “conversor” capaz de receber um elemento léxico, independente da linguagem equivalente a este estilo e apresentar como saída um outro normalizado. Este elemento obtido da conversão deve pertencer a uma linguagem subjacente, em que todo e qualquer *token* desta linguagem possui um equivalente nas demais e vice-versa (devem manter uma relação biunívoca). As linguagens disponíveis para uso do usuário seriam apenas representações diferentes de uma única linguagem subjacente.

Uma implementação literal deste conceito permite que o usuário componha uma “instrução” selecionando uma operação de um menu, fornecendo um dos parâmetros usando o mouse e o outro através de linguagem de comandos. É importante observar que isto não é problema, pois cada um destes *tokens* tem um representante único em uma linguagem subjacente. Todo este tratamento fica contido no nível léxico. O nível sintático desconhece a origem da entrada obtida.

---

<sup>3</sup>*Adaptable User Interface*. Termo cunhado por Kantorowitz e Sudarsky[KS89].

### 3.5.4 Metodologias

O desenvolvimento de sistemas interativos deve dar atenção para a interface logo nos primeiros estágios de desenvolvimento, ao contrário de considerá-la apenas quando um sistema subjacente estiver pronto. As preferências do usuário devem ter prioridade sobre as considerações do ponto de vista do sistema. A metodologia *User Software Engineering* (USE) contempla tais questões. USE[WPSK86] é uma metodologia suportada por ferramentas gráficas automatizadas para o desenvolvimento sistemático de sistemas interativos. Prevê o envolvimento do usuário nos primeiros estágios do processo de desenvolvimento, e o uso de protótipos da interface criados e modificados rapidamente. Isto é similar ao que é defendido pelo ciclo de vida estrela (§3.5.7).

### 3.5.5 Portabilidade

Embora haja consenso acerca de características gerais em interfaces gráficas (“window”, “button”, “scrollbar”, e outras), detalhes de implementação para programação dessas características diferem entre plataformas distintas. Cada ambiente tem suas próprias “bibliotecas” e estruturas de dados. Ajustar essas diferenças não se faz simplesmente alterando chamadas a sub-rotinas, algumas vezes as arquiteturas internas são incompatíveis. Andersen e Sherwood[AS91] caracterizam esse problema.

No projeto desenvolvido tentou-se manter uma separação de aspectos dependentes de ambiente, daqueles independentes. Não é uma solução, mas elemento essencial para criar versões para outras plataformas.

No desenvolvimento realizado utilizou-se do conceito de herança para isolar, em níveis hierárquicos distintos, aspectos dependentes e independentes.

### 3.5.6 Protótipos

Protótipos são elementos essenciais no projeto de uma interface[Cou85, Nie92]. São eles que permitem a avaliação de um produto antes que recursos tenham sido despendidos em vão. A necessidade de construir protótipos é oriunda da carência de princípios que garantam um resultado satisfatório de projeto (veja pág. 51). Mesmo os princípios existentes são difíceis de serem seguidos. Em [HH89a] é comentado exaustivamente a necessidade de protótipos para o projeto de interfaces. O modelo estrela para o ciclo de vida de uma interface destaca esta necessidade (veja figura 3.8).

Draper[DN85] ressalta que modificações na especificação são comuns no projeto de interfaces. Afirma que as modificações são frutos do pouco conhecimento acerca de bons princípios de projeto: especialmente os quantitativos, que praticamente inexistem; já os qualitativos são mais conhecidos.



### 3.5.7 Ciclo de vida

Existem várias pessoas envolvidas no desenvolvimento de interfaces. O projetista da interface define a interface (veja figura 1.2). Ele é responsável pelo projeto. Em [Nie92] e [BC91] é comentado em mais detalhes o projeto de interfaces. Este projetista deve ter conhecimentos acerca dos custos de implementação e manutenção. O projetista de software de interface é responsável pela implementação da interface definida pelo projetista da interface.

Função	Descrição
<i>apague nodo estr mod</i>	Apaga nodo identificado por <i>nodo</i> , da estrutura <i>estr</i> , modalidade <i>mod</i> . O nodo removido fica em <i>buffer</i> temporário.
<i>copie estr<sub>1</sub> estr<sub>2</sub> mod</i>	Copia nodo identificado por <i>nodo</i> da estrutura <i>estr<sub>1</sub></i> para estrutura <i>estr<sub>2</sub></i> na modalidade de cópia <i>mod</i> .
<i>crie nodo</i>	Cria e nomeia um nodo da estrutura.
<i>insira nodo estr parentesco</i>	Insera nodo identificado por <i>nodo</i> na estrutura identificada por <i>estr</i> , na posição identificada por <i>parentesco</i> .
<i>mostre nodo estr</i>	Mostra os valores dos atributos do nodo identificado por <i>nodo</i> na estrutura <i>estr</i> .

Tabela 3.1: Operações básicas do módulo EDITOR.

Projetar é o processo que progressivamente transforma os requisitos (definição de requisitos) de um sistema através de estágios intermediários até a identificação do que deve ser implementado (especificações). No caso específico de uma interface projetar significa derivar a interface com a qual o usuário irá interagir. A atividade de projeto, realizada pelo projetista da interface, compreende as fases de especificação (definição de requisitos) e especificação. Estas fases são comentadas abaixo.

A tabela 3.1 contém operações básicas que um sistema (§4.1) deve realizar através de um de seus módulos (EDITOR). Ela mostra a especificação de tarefas fruto de uma das primeiras atividades de projeto: análise de tarefas. Baseado nesta especificação o projetista da interface sucessivamente refina a especificação. Para o função *crie*, p.ex., foi decidido que o usuário poderia clicar sobre um ícone, selecionar uma opção de menu, ou fornecer o comando “*crie*” via linguagem de comandos para selecionar esta operação. O argumento *nodo* pode ser fornecido através de linguagem de comandos ou uso do mouse.

A atividade descrita acima é de responsabilidade do projetista da interface que deve ter conhecimentos em fatores humanos. Este indivíduo estabelece o *look and feel* da interface e é responsável por garantir que o projeto atinja as qualidades desejáveis em uma interface.

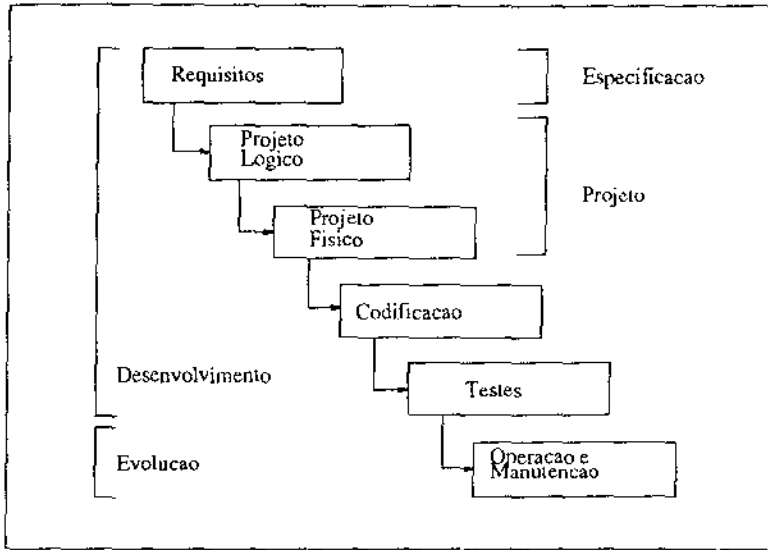


Figura 3.7: Modelo em Cascata do ciclo de vida de *software*

Tais qualidades e princípios necessários para alcançá-los foram vistos na seção anterior.

A atividade do projetista da interface está além do escopo deste trabalho. O desenvolvimento proposto apenas considera fatores humanos e questões psicológicas como preconiza a literatura (§1.3).

A atividade de maior interesse aqui é realizada pelo projetista de software da interface. Ele é encarregado de implementar a especificação realizada pelo projetista da interface (responsável pelo projeto). Veja-se §3.5.7 sobre funções envolvidas no desenvolvimento de interfaces.

Do ponto de vista de software, um sistema passa por várias fases até que seja implementado, mesmo após a entrega do sistema ele sofre modificações. Estas várias fases constituem o que se entende por ciclo de vida de um sistema. Um modelo tradicional para ciclo de vida é o Modelo em Cascata (*Waterfall*) visto na figura 3.7.

Neste modelo cada fase produz resultados que fluem para o próximo estágio, idealmente isto ocorre de forma ordenada e linear. Para mais detalhes sobre este modelo veja [GJM91].

A idéia de que o ciclo de vida tradicional (*Waterfall*) é inadequado para o desenvolvimento de sistemas interativos origina-se da natureza *tentativa e erro* do desenvolvimento de interfaces. Trata-se de processo cíclico [CH88, Shn87]. Projetistas de software ainda não conseguem prever as conseqüências de decisões de projeto no uso de uma sistema [Abo91, pág. 2]. Newman e Sproull [NS79, pág. 443] também já haviam ressaltado a incapacidade em se prever o desempenho de uma interface. Em conseqüência, rapidez e facilidade de modificação são indispensáveis. Boehm [Boe88] também afirma que o modelo *Waterfall* não é apropriado para algumas classes de software, particularmente os sistemas interativos. Ainda não se co-

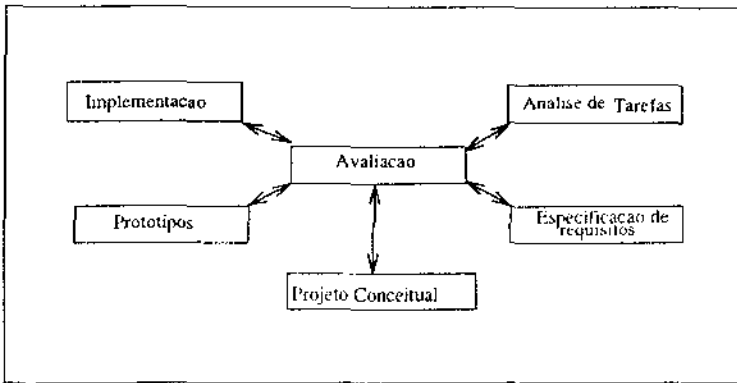


Figura 3.8: Ciclo de vida estrela para o desenvolvimento de interfaces.

nhece princípios de projeto que, uma vez seguidos, garantam resultados satisfatórios. Desse modo, uma abordagem evolucionária com a construção de protótipos — algo mais próximo da espiral de Boehm[Boe88] — é frequentemente mais efetiva[WPSK86].

Recentemente em [HH89b] foi proposto o modelo estrela (figura 3.8). O objetivo deste modelo é fornecer uma estrutura de desenvolvimento mais adequada para interfaces. Foi resultado de observações empíricas. As observações mostram que o ciclo de vida tradicional (*Waterfall*), *top-down*, isola as atividades de requisitos, projeto, implementação e teste; impor este paradigma conduz a interfaces de qualidade pobre. Uma das bases do modelo está no que foi chamado de *alternating waves*. Refere-se ao fato de que o projeto de interfaces não é uma atividade nem *top-down*, nem *bottom-up*, mas uma mistura. O projetista da interface pode alternar entre eles repetidas vezes.

As primeiras atividades de projeto são *bottom-up*, baseadas em cenários e frequentemente ampliadas com representações da evolução do diálogo como diagramas de estados. Estão relacionadas à visão do usuário. Atividades seguintes seriam *top-down*, relacionadas à visão do sistema.

No centro (figura 3.8) encontra-se a avaliação. Qualquer que seja a atividade em desenvolvimento, ela pode ser rapidamente substituída por outra e se reiniciar. Este modelo contempla especificamente o desenvolvimento da interface e difere, portanto, da espiral de Boehm. Outra diferença está no uso de protótipos e testes. A espiral de Boehm contempla tais atividades em ciclos completos de desenvolvimento. Hartson e Hix[HH89b] afirmam que pequenos ciclos são importantes para os testes com interfaces e são facilmente permitidos com o modelo estrela.

### Definição de requisitos (atividade inicial de projeto)

Envolve a definição de um problema, modelagem do usuário e análise de tarefas. O modelo do usuário contém suposições acerca do entendimento do domínio do usuário, suas idiossincrasias

e conhecimentos de computação.

### Especificações (atividade final de projeto)

Nesta fase o projetista de interface especifica a interface com a qual o usuário irá interagir. Os objetos de interação, tais como janelas, comandos, e menus são determinados. E definição léxica da interface (cores, posições, nomes de comandos e assim por diante). Estas especificações buscam satisfazer os princípios vistos anteriormente.

A especificação de uma interface deve compreender o uso de várias linguagens focalizadas em aspectos distintos [HH89a, Gre85]. Neste trabalho interessa a especificação de controle de diálogo. A figura 3.4 na página 49 mostra uma arquitetura de software organizada em várias camadas. A camada do controlador de diálogo é a responsável pelo controle de diálogo. Entre as notações mais comentadas para registrar este controle estão os Diagramas de Transição de Estados (DTEs). Estes diagramas apresentam inconvenientes que foram eliminados em uma de suas extensões: os Estadogramas.

Cada camada tem finalidades distintas das demais e pode-se pensar em uma linguagem para descrever cada uma delas. Contudo, o projetista da interface não vê um sistema interativo desta forma. Esta concepção é eminentemente de implementação. Sua especificação preocupa-se com fatores humanos que são independentes de arquiteturas de software. O que é importante destacar aqui é a diferença de conhecimento entre especialistas em áreas distintas.

### Propriedades de uma técnica de especificação

Jacob em [Jac83] fornece uma lista de propriedades desejáveis em uma técnica para especificação de interfaces. No capítulo 5 são feitos comentários relacionando estas propriedades e os Estadogramas.

Propriedades:

- *A especificação deve ser de fácil compreensão e, sobretudo, mais fácil de ser produzida do que o software que a implementa.*
- *Deve ser precisa.* Não podem existir ambigüidades ou dúvidas acerca do comportamento do sistema para cada possível entrada do usuário.
- *Deve facilitar a verificação da consistência.*
- *Deve ser poderosa o suficiente para expressar sistemas de comportamento não trivial com um mínimo de complexidade.*
- *A implementação deve ser independente da especificação, ou seja, deve haver uma separação entre "o que" e "como".*

- Deve permitir a construção de protótipos diretamente da especificação.
- A estrutura da especificação deve ser próxima do modelo do usuário (veja §3.3).

Especificações de diálogo são independentes de hardware, assim pode-se usar plataformas baratas para permitir a construção de protótipos do diálogo.

### Dificuldades com a especificação de diálogo

Técnicas usadas para especificação de interfaces ainda apresentam problemas. Green[Gre86] identifica três grupos para caracterizar modelos de diálogo: gramáticas, redes de transição e eventos. Abaixo seguem dificuldades encontradas em cada um destes grupos.

- A especificação de diálogo deve permitir a construção rápida de protótipos, devido ao caráter iterativo de projeto. As notações para este uso, portanto, necessitam ser formais para permitir a geração automática de código ou interpretação da especificação.
- Representações baseadas em gramáticas são impróprias para especificar certos estilos de interfaces como manipulação direta. A maior dificuldade reside na interpretação humana destas descrições. Gramáticas cederam espaço para as redes de transição, muito mais intuitivas e fáceis de serem construídas.
- O uso de redes de transição não permite uma composição hierárquica, o que conduz a extensas e complicadas especificações. Outro inconveniente reside na sua característica eminentemente seqüencial, o que torna inviável a especificação de diálogo *multithread* (§5.4.2), por exemplo.

O modelo de eventos surge na tentativa de expressar paralelismo. Neste caso tem-se um conjunto de regras na forma “se COND então AÇÃO” equivalente à interface. Regras cuja condição seja verdadeira provocam a execução das ações correspondentes. Não há transferência explícita de controle, isto obriga monitorar constantemente as regras, que podem ser em grande número. Isto também dificulta a especificação, pois exige soluções paralelas para os problemas. Outra dificuldade reside na inexistência de uma hierarquia de especificação.

Myers[Mye89] ainda cita que é muito difícil criar código correto devido ao controle descentralizado. Pequenas mudanças podem afetar todas as outras partes, além de inviabilizar a compreensão a medida que o código cresce.

- A especificação de uma interface envolve a representação de aspectos visuais. BNF (textual) e diagramas de transição e estados (gráfica) são próprios para a representação de relacionamentos gramaticais (por exemplo: seqüência lógica dos comandos e seus parâmetros). Observando especificações feitas com as técnicas acima não se sabe, por exemplo, como serão fornecidas as entradas por parte do usuário, ou seja, não fica

estabelecido o estilo de diálogo. Contudo, isto pode ser útil para uma análise do diálogo além de sua aparência. Jacob[Jac83] afirma que diagramas são preferíveis à notação BNF. Estes diagramas, contudo, também possuem problemas como viu-se em §2.7.1.

Os problemas conhecidos dessa abordagem são as conexões entre a interface e a aplicação realizadas através de variáveis globais, bem como a necessidade de explicitar para todos os estados arcos para todas as possíveis entradas e de explicitar todos os comandos, inclusive *help* e *undo*[Myc89]. Outra dificuldade comumente citada é a incapacidade em representar interfaces em que o usuário pode operar em múltiplos objetos concorrentemente.

- Wasserman[Was85] fornece uma extensão de diagramas de transição de estados com o intuito de satisfazer alguns princípios que considera essenciais em uma notação apropriada para especificação de interfaces. Um deles é completitude, ou seja, a notação deve ser suficiente para representar todos os aspectos de uma interface. No entanto, conforme Harton e Ilix[HII89a], o problema de completitude em representações de interface permanece sem solução. Não somente uma técnica seria suficiente, mas um conjunto destas se faria necessário para registrar o comportamento, a estrutura e a representação de aspectos visíveis e não visíveis de uma interface.

### Implementação

A interface fornece ao usuário uma máquina abstrata com a qual ele pode interagir. Uma maneira de estruturar o software é como uma hierarquia de máquinas abstratas, cada uma fornecendo certos serviços para as camadas acima e requisitando serviços das camadas abaixo. Esta divisão em máquinas abstratas fornece um mecanismo para proteger uma camada de modificações abaixo desta camada. Isto também conduz a um maior grau de portabilidade. Os princípios de engenharia de software subjacente a estas máquinas abstratas são modularidade e *divisão de responsabilidades* (veja [GJM91]).

Em §2.3 é estabelecida uma taxonomia para estas máquinas abstratas.

## 3.6 Ferramentas

Viu-se (capítulo 1) que um especialista em fatores humanos está mais habilitado do que um programador para definir (realizar as atividades de projeto) uma interface. Surge um problema diante desta realidade: o que este especialista deve utilizar para descrever uma interface? Naturalmente, a linguagem C pode ser utilizada para esta especificação, mas é de muito baixo nível para ser útil e é de uso exclusivo por programadores. Existem linguagens de propósitos específicos para a especificação de interfaces. Geralmente uma linguagem possui uma ferramenta subjacente, que permite contemplar a construção rápida de protótipos ou a

Metodologia	Exemplo	Vantagens	Desvantagens
Programação baseada em exemplo	Peridot e Lapidary	Facilidade de usar e aprendizagem rápida	Mecanismo de inferência pode não ser confiável. Abordagem baseada em inferência pode levar mais tempo que métodos de especificação diretos
Especificação por restrição gráfica	ThingLab e Symbolics Freme-Up	Flexível e em geral declarativa. Abordagem promissora para desenvolvimento de novas formas de interação.	Pode ser computacionalmente inviável. O relacionamento de restrições pode ser complexo e difícil de entender.
Abstração de comportamento	GSS e Garnet	Facilidade de usar. Comportamentos podem ser facilmente combinados para construir interações sofisticadas.	Tipos de comportamentos limitados. Tempo de aprendizagem pode ser longo para sistemas com grande número de comportamentos.
Sistemas de diagramas	Interpretador de diagramas de estados	Fácil de entender. Modelo intuitivo para compreensão do fluxo de informação.	Interpretação de grandes e complexos diagramas é difícil. Dificuldade para aplicar tarefas de desenvolvimento necessárias.
Ambientes de objetos de alto nível	Trillium e NeXT Interface Builder	Facilidade de uso. Aprendizagem rápida	Relativamente inflexível. Restrito a estilos de interação específicos e capacidade de apresentação limitada.

Tabela 3.2: Vantagens e desvantagens de metodologias para projeto de interface.

sua implementação. Em alguns casos, ambas as opções estão disponíveis refletindo o projeto e a implementação.

As linguagens de propósito específico podem ser vistas como parte das técnicas disponíveis para especificação de interfaces. DeSoi e Lively [DL91] descrevem e analisam várias técnicas para uso por não programadores. A tabela 3.2 (retirada de [DL91]) descreve concisamente algumas características destas técnicas.

Nesta tabela são contemplados os diagramas de transição de estados, classe onde se “encaixam” os Estadogramas. DeSoi e Lively afirmam que para estes diagramas as ações realizadas como consequência das entradas do usuário devem ser programadas em uma linguagem convencional. Os diagramas são usados para a especificação de diálogo e/ou como mecanismo para conectar a interface à aplicação (funcionalidade). Neste trabalho usa-se os Estadogramas tanto para esta conexão quanto para a especificação de diálogo. É importante observar que as observações feitas por DeSoi e Lively são consistentes com a taxonomia estabelecida anteriormente para os sistemas reativos (§2.3). Nesta taxonomia viu-se que Estadogramas são utilizados para a especificação do núcleo reativo.

*Toolkit* é uma biblioteca que fornece facilidades para a construção de menus, *buttons*, e *scroll bars*. O usuário de um *toolkit* é um programador. *Toolkits* fornecem facilidades relacionadas aos objetos de interação (§3.4).

Um construtor de interface (*interface builder*) é uma ferramenta gráfica que auxilia o programador a criar caixas de diálogo, menus e outros controles. Está limitado a partes estáticas de uma interface.

Quando várias ferramentas estão integradas e visam o desenvolvimento de interfaces, denomina-se este conjunto de ferramentas de UIMS.

### UIMS (*User Interface Management System*)

Esta seção cobre de forma resumida alguns aspectos destes sistemas. Mais informações podem ser obtidas em [Hix90], que discorre sobre as gerações de tais sistemas. Em [HH89b] é fornecida uma visão destes sistemas envolvendo uma variada faixa de questões. Atualmente a utilização e funcionalidade destes sistemas ainda pode ser melhorada [MR92]. Para questões específicas de desenvolvimento destes sistemas veja [TB85].

---

---

**UIMS:** sistema voltado para o desenvolvimento e execução de interfaces entre homem e computador. Ajudam na especificação, projeto, construção de protótipos, implementação, execução, avaliação, modificação e manutenção de interfaces, conforme Hix[Hix90].

---

---

Hix ainda afirma que o papel de um projetista de interface é atualmente aceito como atividade que não exige conhecimentos de programação.

São quatro as gerações:

- Primeira geração: Compreende construtores de protótipos e gerenciadores de tela. Os construtores geram apenas uma “maquete” da interface para uso em simulação. Os gerenciadores eram empregados tipicamente para produzir as telas da interface. Permitia um conjunto restrito de estilos de interação: menus e *form-fill*. As interfaces produzidas eram especificadas em BNF (*Backus-Naur form*). Esta primeira geração é voltada para uso por programadores e concentra-se na implementação de interfaces.
- Segunda geração: Está voltada para o suporte fornecido às interfaces em tempo de execução. Projetados para uso por programadores. Pouca ênfase é dada no projeto, fatores humanos e o usuário. Em vez das gramáticas (BNF) usa-se diagramas de transição de estados.
- Terceira geração: Permite ao projetista usufruir de conceitos como manipulação direta. Enfatizam as atividades de projeto bem como suporte em tempo de execução.



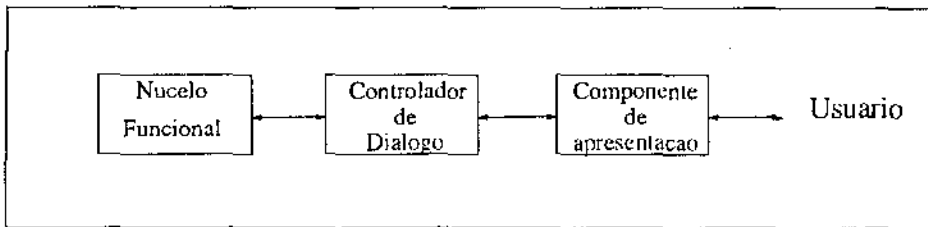


Figura 3.9: Participantes de uma interação homem-computador.

Apresenta um aumento em funcionalidade e facilidade de uso com relação às anteriores.

- Quarta geração: Destaca-se por fornecer maior funcionalidade e facilidade no uso. Marca o início do uso de inteligência artificial através de bases de conhecimento e sistemas especialistas.

### 3.7 Requisitos para o controlador do diálogo

Na vida real uma conversação entre vários participantes é conduzida por um mediador (coordenador) ou distribuída entre os participantes. Em geral o controle de diálogo alterna entre esses dois extremos. Esta seção discorre sobre os requisitos para um controlador de diálogo de sistemas interativos.

Conforme [BC91], a interação entre o homem e o computador pode ser observada em vários níveis de abstração. Um deles mostra a interação como a cooperação entre quatro participantes (veja figura 3.9).

O controle de diálogo define a política de conversação entre os participantes (usuário e sistema interativo) de um diálogo. Em alguns instantes a participação dos elementos envolvidos pode se sobrepor. Tanner e Buxton[TB85] classificam o fluxo de controle no interior de um sistema interativo em externo, interno e misto. Se o controle reside na aplicação e ao usuário resta apenas responder às indagações tem-se um controle interno. Por outro lado, se o controle é guiado pelas ações do usuário ele reside no controlador do diálogo e é dito externo. Neste último caso a aplicação é vista como um servidor semântico. O modelo misto assume que o controle permuta entre estes dois elementos, ou seja, ora reside no controlador e ora na aplicação.

O modelo misto é útil quando, por exemplo, tem-se tanto atividades que devem ser conduzidas pelo usuário quanto aquelas em que ele é essencialmente um observador. A aplicação pode estar processando e continuamente enviando mensagens ao controlador para que o componente de apresentação possa refletir as mudanças. Enquanto isto, é natural que o usuário possa alterar a seqüência de execução ou mesmo interrompê-la. Convém observar que isto exige um acesso concorrente ao controlador, pois tanto a aplicação quanto ações do usuário podem ocorrer simultaneamente e devem ser tratadas de acordo.

## 3.8 Interfaces e o Paradigma de Objetos

O paradigma de objetos é freqüentemente relacionado a vários aspectos de interfaces. Esta seção mostra um pouco do relacionamento entre esta abordagem para a solução de problemas e o problema da construção de interfaces. Conforme Khoshafian[KA90], o paradigma é explorado para o desenvolvimento (projeto e implementação), apresentação e integração de interfaces.

1. **Projeto e implementação.** Devido à quantidade de rotinas e o grande número de parâmetros que geralmente apresentam, um programador não usa diretamente a interface de programação de um sistema de janelas. Ele faz uso de uma camada orientada a objeto sobre a interface de programação. Esta camada equivale aos objetos de interação (§3.4). Como exemplos existem MacApp para o Macintosh e Actor para Windows. Uma descrição destas duas camadas pode ser vista em [KA90].

A camada orientada a objetos fornece uma hierarquia de classes predefinidas. Elas reduzem a quantidade de código a ser gerada pelo programador e fornecem consistência (§3.5.2) à interface. Encapsulam o comportamento e a apresentação de elementos comuns como menus, janelas, e ícones. O mecanismo de herança pode ser utilizado para herdar comportamento e estrutura de superclasses. Os elementos herdados podem ser redefinidos ou estendidos. Mais detalhes de que tipo de facilidades e como podem ser usadas são apresentados na descrição do *InterViews*<sup>4</sup> em [LCV].

2. **Apresentação.** Nas modernas interfaces o usuário geralmente seleciona ou move ícones e objetos sobre a tela. Estes objetos e ícones são representações gráficas de conceitos do mundo real (metáfora, §3.5.2) e, portanto, reagem a mensagens recebidas bem como comunicam-se com outros objetos. Nestes sistemas, p. ex., arquivos podem ser manipulados como objetos. Para remover um objeto a mensagem remove é enviada ao objeto correspondente.
3. **Integração.** Este emprego é mais sutil que os demais. Em vez de diretamente usufruir do paradigma de objetos, faz uso de conceitos como objeto complexo e identidade de objeto. Análogo à confecção de um objeto complexo, uma aplicação complexa pode ser obtida pela integração de informação de um conjunto de aplicações. O usuário pode desenvolver um relatório (objeto complexo) em que gráficos foram gerados de resultados obtidos de uma planilha eletrônica, que obteve informações de um banco de dados. Convém notar que os dados não são simplesmente transferidos de uma aplicação para outra. Uma alteração em um objeto causa a alteração do estado do objeto complexo.

---

<sup>4</sup>Camada orientada a objeto sobre o sistema *X Window*. Sistemas de janela são cobertos na página 43.

### 3.9 Resumo

O objetivo deste capítulo é estabelecer um conjunto mínimo de informações para o desenvolvimento de uma interface, e identificar “elementos” que de alguma forma ou de outra relacionam-se com o comportamento de interfaces. O texto não é introdutório, mas traz referências que podem ser consultadas para a obtenção de maiores detalhes. Foram vistos que estilos de interação são relacionados com o comportamento de interface (*feel*), bem como exigências oriundas da característica iterativa do desenvolvimento de interfaces, ciclo de vida e ferramentas utilizadas para implementação. Tratam-se de informações essenciais para identificar o atual estado da arte no desenvolvimento de interfaces e os problemas existentes.

O texto está dirigido para o problema deste trabalho e agrega várias informações que, embora já publicadas, encontram-se dispersas na literatura.

O capítulo anterior concentrou-se nos Estadogramas e sugestões propostas neste trabalho. As sugestões foram baseadas, em grande parte, nas informações contidas neste capítulo, que serviu de orientação e fundamento para o trabalho. O próximo capítulo discorre sobre o desenvolvimento realizado. Nele são descritos o sistema para o qual uma interface particular foi construída, detalhes da interface e do seu desenvolvimento.

## Capítulo 4

# Desenvolvendo uma Interface

*“A construção de protótipos é uma técnica,  
não é uma panacéia.  
Ela só acelera o processo,  
não assegura a alta qualidade de um projeto.”*  
Wilson e Rosenberg[WC91]

Neste trabalho é previsto o desenvolvimento de uma interface para que Estadogramas possam ser analisados como notação e linguagem, para descrever e implementar o controle de diálogo de interfaces.

No capítulo anterior foram apresentados algumas diretrizes para conduzir desenvolvimento de interfaces. Neste capítulo é definido o sistema para o qual o protótipo de uma interface particular está sendo desenvolvido; o protótipo em desenvolvimento e o ambiente onde ocorre a implementação. O capítulo também descreve o que estava implementado do sistema e da interface (protótipos) até o momento em que foi concluído este texto.

---

### 4.1 Um Sistema

Uma descrição completa do sistema para o qual a interface foi desenvolvida encontra-se em [Bac]. [Bac] é uma proposta para um ambiente de experimentação musical. Esta seção fornece apenas uma visão geral do sistema. As omissões, contudo, não prejudicam o propósito deste trabalho.

O **Micromundo Musical (MM)**, nome dado ao ambiente de experimentação musical, substitui as partituras (notação tradicional) por uma representação em forma de árvore. As estruturas de uma música são ressaltadas pela hierarquia representada na árvore. O músico ou usuário interage com uma árvore, em vez das convencionais partituras.

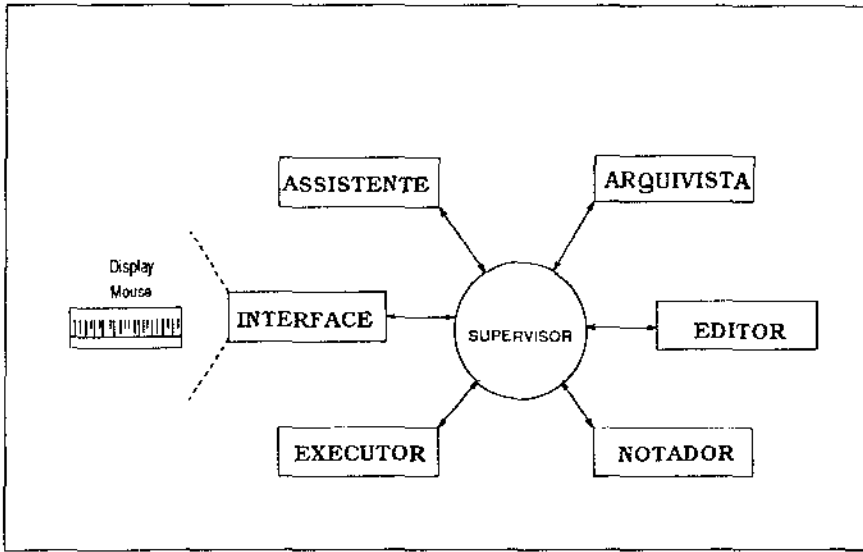


Figura 4.1: Arquitetura do Micromundo Musical.

Em [Bac] são exemplificados vários sistemas musicais. A representação musical em forma de árvores já foi utilizada anteriormente em outros sistemas.

A representação interna de uma composição musical pode ser vista na figura 4.2. Esta estrutura também é exibida para o usuário do MM em forma de árvore. A figura 4.8 exemplifica um instante da interação com o protótipo do MM. Assim, a árvore na figura 4.8 é a apresentação de uma árvore interna que representa uma composição, exemplificada na figura 4.2.

Retornando à árvore (figura 4.2), vê-se a presença de nodos terminais e intermediários. Cada nodo intermediário possui informações úteis para toda a sub-árvore do qual é pai. Os nodos terminais correspondem a eventos musicais que contêm as notas musicais. Para cada nodo terminal tem-se uma lista que armazena informações correspondentes às notas musicais. As informações são armazenadas em formato MIDI (*Musical Instrument Digital Interface*). MIDI é a especificação de um esquema de comunicação para dispositivos musicais digitalizados. Trata-se da linguagem compreendida pelo teclado (sintetizador), veja-se figura 4.1. Convém ressaltar que as composições não são armazenadas (disco) em forma de ondas, mas em MIDI.

A figura 4.1 representa a visão de um projetista de software do MM. Distinta, naturalmente, da visão do projetista de software para interface (figura 3.4). Ela mostra vários módulos existentes no sistema e como o controle flui entre eles. A seção seguinte descreve sucintamente cada um deles. A tabela 3.1 na página 56 descreve as funções executadas pelo módulo EDITOR.

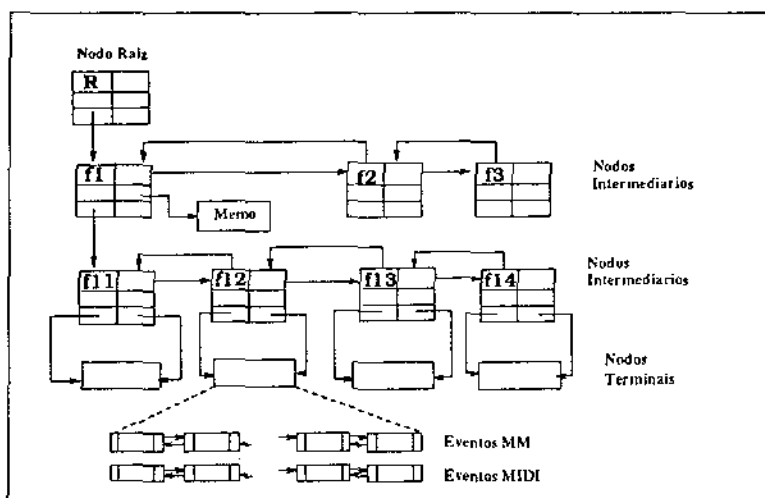


Figura 4.2: Árvore

### Descrição dos módulos do MM

**SUPERVISOR** Responsável pela identificação, validação e distribuição das ativações dos módulos do MM. Essas ativações são mensagens trocadas entre módulos requisitando serviços. O mecanismo de comunicação usado no protótipo é realizado através de chamada de procedimentos. A implementação definitiva deve considerar cada módulo como um processo que opera de forma assíncrona com relação aos demais.

**ASSISTENTE** Trata situações de conflito, ambigüidade ou insuficiência de informações nas ações do usuário; coleta tais ações; realiza registro de contexto e parâmetros de configuração. Este módulo não está completamente construído, mas seus objetivos conduzem ao conceito de interface inteligente.<sup>1</sup>

**ARQUIVISTA** Gerencia os dados musicais (armazenamento e recuperação em bases de dados).

**EDITOR** Manipula os objetos musicais, tanto a nível estrutural quanto sônico.

**NOTADOR** Mantém as estruturas de apresentação internas. A apresentação refere-se ao que o usuário do MM pode perceber visualmente. Este módulo deve gerar a disposição e gerenciamento da apresentação visualizada pelo usuário, ou seja, uma árvore.

**EXECUTOR** Módulo encarregado da sonorização, i.e., gerar e manipular código MIDI.

<sup>1</sup>A interface é vista como intermediário "inteligente" entre agentes que não se conhecem completamente podendo diagnosticar situações de conflito. Este conceito está além do escopo deste trabalho. Recomenda-se *Intelligent Interface Design* de Chignell e Hancock (*Handbook of Human-Computer Interaction*, M. Helander, Elsevier Science, 1988).

## 4.2 Visão geral do desenvolvimento do sistema

Exceto o módulo interface, todos os demais estão totalmente independentes do modo que suas estruturas de dados e funções são expostas para o usuário. Não estão conscientes do meio usado para entrada e saída, bem como a forma específica na qual a informação é transmitida, ou seja, tem-se independência de diálogo (pág. 52).

O módulo NOTADOR não foi implementado conforme a arquitetura descrita e especificada em [Bac]. Nota-se com facilidade a semelhança funcional deste módulo com os aspectos léxicos de um sistema interativo (§3.3). A parte funcional deve ignorar qualquer informação que seja dependente de meios de exibição. Dessa forma, nenhum outro módulo necessita saber da existência do NOTADOR. Suas funções são de competência do módulo INTERFACE.

O sistema evolui paulatinamente com o acréscimo de funções ao protótipo (do sistema) até então obtido. Apenas um subconjunto da funcionalidade prevista está disponível. O desenvolvimento, contudo, prossegue isolado da interface, sem prejudicar o andamento da mesma.

Neste trabalho não se teve particular interesse por metodologias, técnicas e princípios de engenharia de software aplicados ao desenvolvimento do sistema. Convém salientar que o sistema e a interface estão sendo desenvolvidos por pessoas distintas que interagem. Contudo, cada um tem os seus problemas específicos. A interação visa determinar, principalmente, o protocolo de comunicação entre estes módulos.

A forma, notações, diagramas eventualmente utilizados no desenvolvimento do sistema, portanto, não são encontrados neste trabalho.

## 4.3 Uma Interface

Esta seção fornece uma visão geral da interface desenvolvida do ponto de vista do usuário. São exemplificados vários cenários relacionados ao protótipo (da interface) até então obtido com a descrição do controle do diálogo em Estadogramas equivalentes.

Identificar uma interface representativa é atitude razoável para os propósitos deste trabalho. O sistema, embora relacionado à interface, está além do contexto deste trabalho e poderia ser escolhido circunstancialmente. A preocupação existiu quanto a uma interface que não envolvesse pontos sujeitos a discussão ou que não fosse abrangente o suficiente para satisfazer tais propósitos. A descrição da interface, contudo, mostra que ela não é trivial.

A primeira tela que o usuário tem contato é a da figura 4.4. A caixa de diálogo no centro mostra algumas informações do sistema. Para que o usuário possa prosseguir é necessário que pressione o botão "OK."

O estado Versão (§4.3) capta o instante em que a identificação (nome, data da criação,

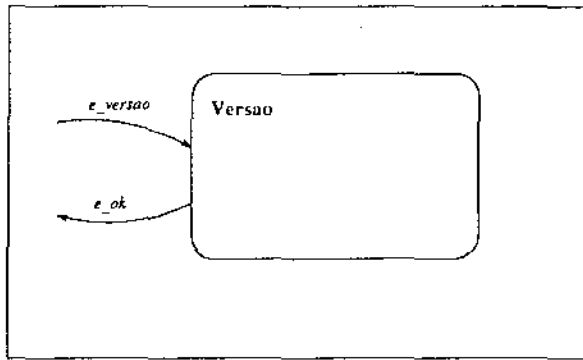


Figura 4.3: Um simples estado.

objetivos e assim por diante) do sistema aparece em uma caixa de diálogo. Equivale ao item de menu “About” comum de muitos sistemas interativos. Quando esta caixa de diálogo estiver ativa nenhuma outra informação é obtida do usuário enquanto não confirmar através do evento *e\_ok*. Ou seja, antes de prosseguir operando o sistema é necessário uma confirmação. Gerar este evento é a única forma de sair deste estado e voltar a operar normalmente.

A interface permite a coexistência de vários estilos de diálogo (§3.2); a figura 4.6 exibe formas alternativas de se interagir com o sistema. Para finalizar a operação do sistema o usuário pode usar a linguagem de comando e digitar o comando “fim.” Uma caixa de diálogo então se abre para que o usuário confirme sua intenção antes de ser executada. Outra forma seria através da opção “fim” do menu. Uma terceira opção permite o mesmo efeito pressionando-se a tecla “ESC” (neste caso trata-se de um atalho, i.e., *shortcut* e não um estilo). Uma outra opção é através de manipulação direta. O usuário pode deslocar o cursor (com uso do mouse, por exemplo) até o ícone que sugere a opção de saída (canto inferior direito). Se o botão esquerdo do mouse é clicado, neste instante, ocorre o mesmo efeito dos casos anteriores. Estas ações do usuário geram eventos físicos que são convertidos no evento lógico *fim*. Neste instante o estado do sistema altera-se de **Normal** para **Confirma Fim**, conforme figura 4.5. A entrada neste último estado gera a caixa de diálogo vista na figura 4.8. A saída remove-a. Para permitir acesso à funcionalidade através de vários estilos de interação existe o conceito de interface flexível (§3.5.3).

O estado **Operação** (figura 4.5) descreve o andamento da execução do sistema. **Normal** reflete o sistema em execução, contrasta com **Inativo**, estado utilizado para representar a situação do software fora de execução (repousa em algum dispositivo de armazenamento secundário). O estado inicial do sistema é o estado **Inativo**. Se o usuário confirmar sua intenção pressionando o botão YES, o evento lógico correspondente (*OK*) é gerado e finaliza-se a operação. Caso contrário, *No\_ok* é gerado e o sistema continua operando normalmente.

A interface possui recursos nem sempre encontrados em outras tradicionais.<sup>2</sup> Por exem-

<sup>2</sup>Considere sistemas interativos onde o usuário pode vasculhar informações geralmente em tabelas, ou observá-las através de gráficos, atualizar valores usando *form-fill* e outros onde geralmente manipulação



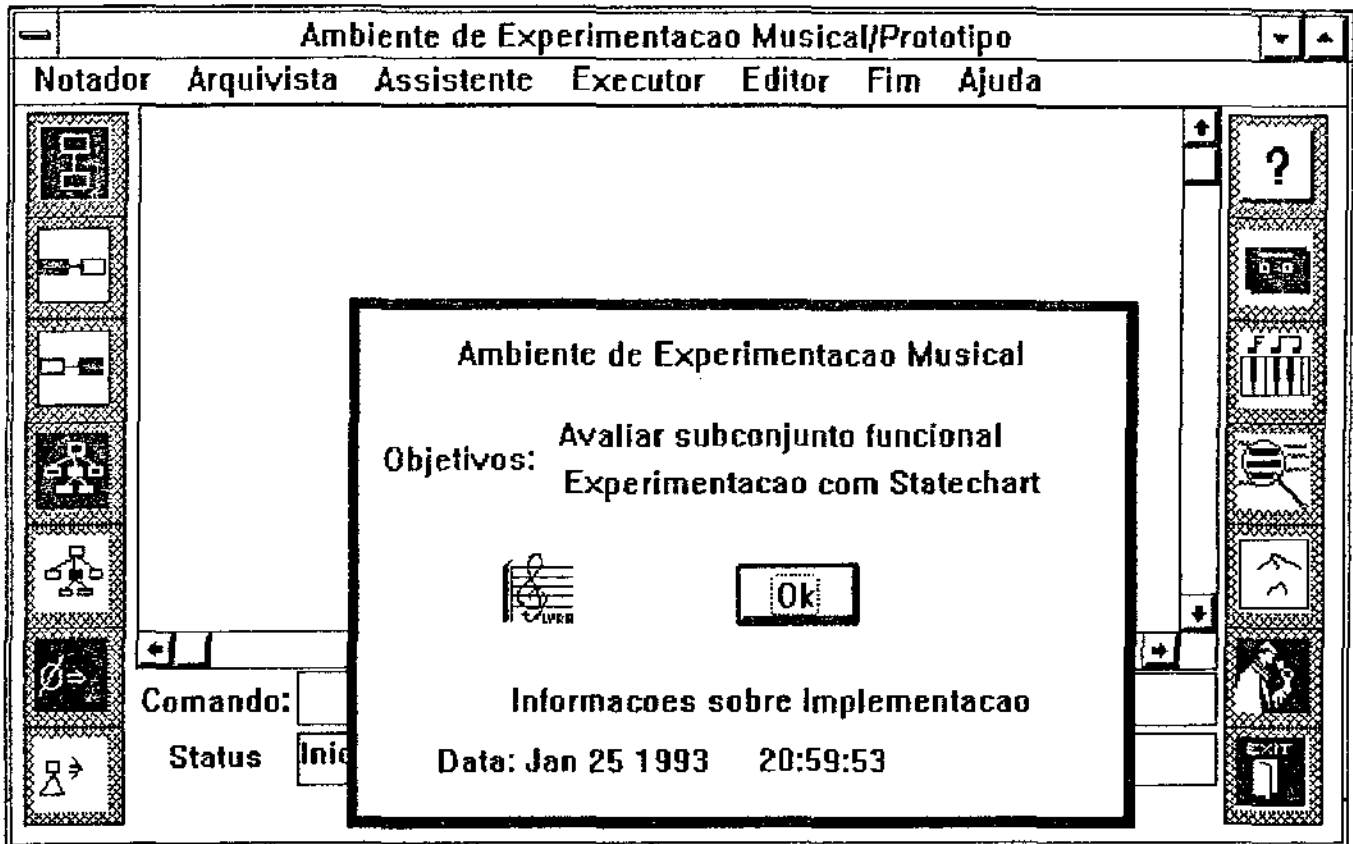


Figura 4.4: A primeira tela

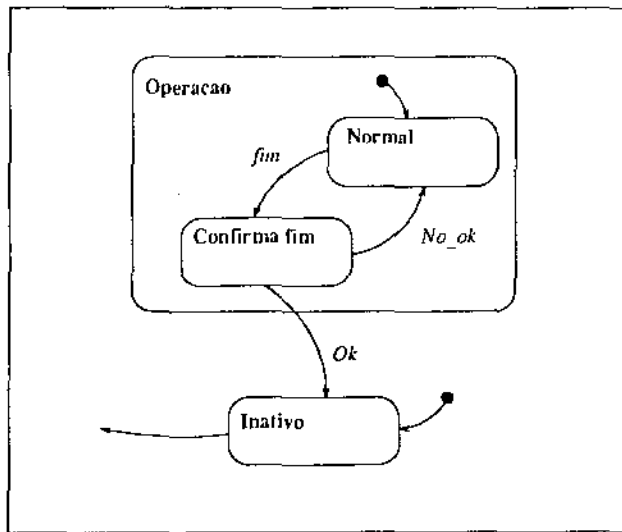


Figura 4.5: Estado Operação.

plô, para permitir que o usuário trabalhe com o sistema foi necessário tornar disponível todas as opções de um editor de árvores (estrutura que representa uma composição). A figura 4.8 mostra um instante de interação com o sistema. Nota-se a existência de várias árvores. Sempre que o usuário passa o cursor sobre um nó, a cor em que é escrito o nome (identificador) do nó é alterada. Quando o cursor abandona os limites do nó, a cor anterior é restaurada. Esta situação é facilmente captada pela descrição na figura 4.9. A figura 4.8 exemplifica um instante da interação em que o cursor está sobre um nó. Cabe ressaltar que imediatamente após o cursor deslocar-se para fora do nó o evento *Outside* é gerado.

Ainda são permitidas operações de *zoom*. Quando o usuário clica o botão da direita do mouse sobre o ícone de *zoom* (quarto ícone da coluna da direita) dispara-se a ação *Zoom\_in()*, se o botão esquerdo é clicado tem-se *Zoom\_out()*. A figura 4.7 capta este comportamento e exemplifica a utilidade de uma das alterações propostas no capítulo 2. Claramente fica registrado que, no instante em que o cursor entrar na superfície do ícone, este último será destacado. O efeito inverso ocorre no momento em que o cursor abandonar a fronteira do ícone. Se o evento *MIDown* causasse uma saída e uma entrada no estado **Highlight** ter-se-ia a realização de duas operações inutilmente.

A interface envolve características comumente encontradas e outras menos freqüentes em interfaces. Ela, portanto, é abrangente e representativa o suficiente para os propósitos deste trabalho. Muitos conceitos utilizados sintetizam o que é apresentado por grande parte das atuais interfaces.

A interface suporta a distinção entre o usuário especialista e o novato (princípio). Por

---

direta não é utilizada, onde princípios de computação gráfica quase não são requeridos, onde o que o usuário vê quase sempre é estático. Enfim, onde o diálogo é simples.

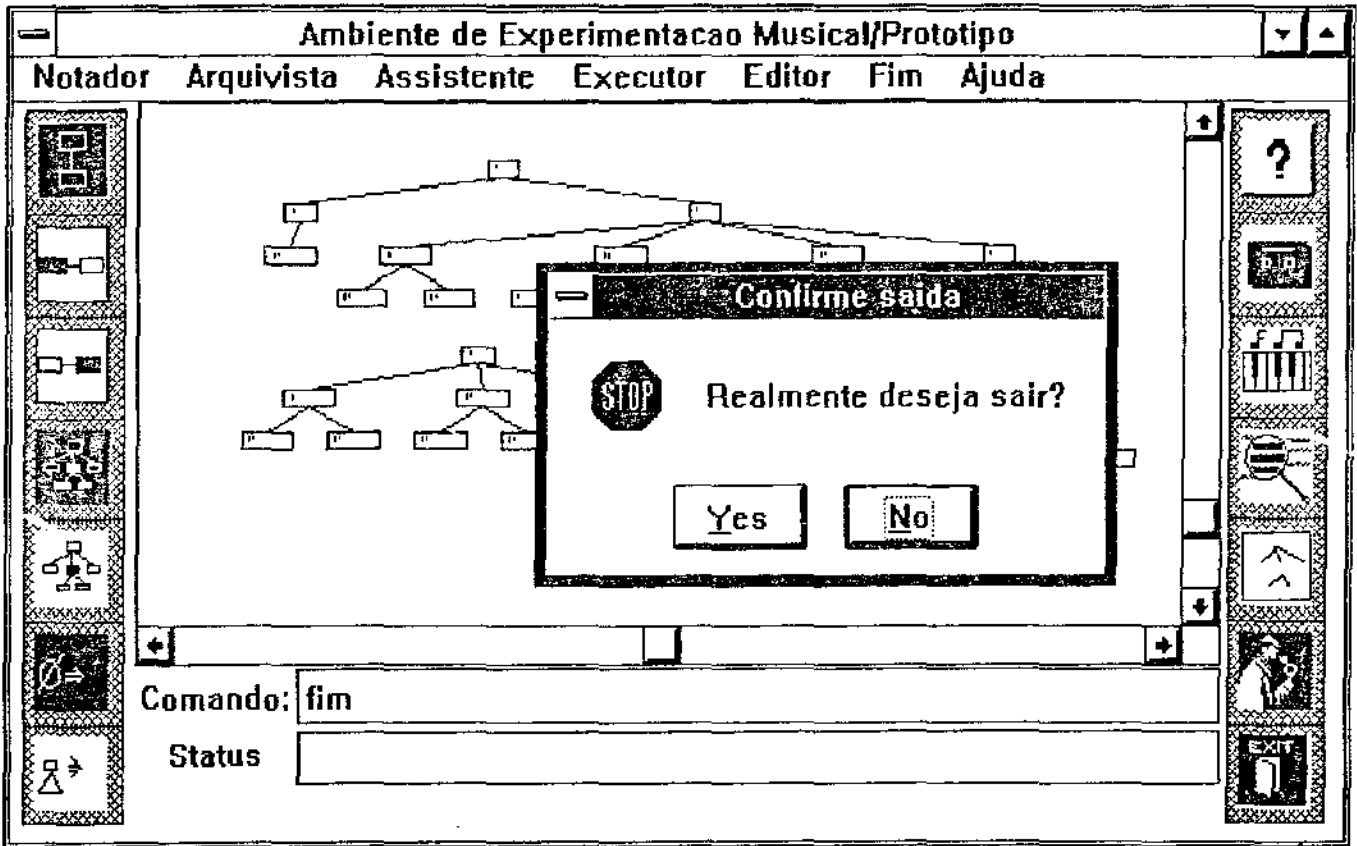


Figura 4.6: Estilos de diálogo disponíveis.

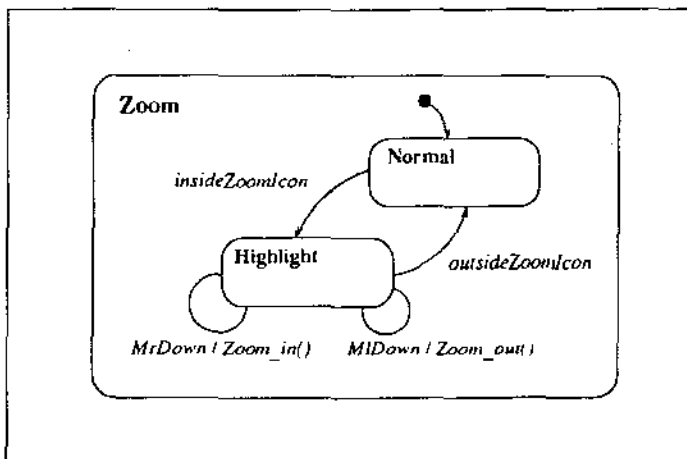


Figura 4.7: Modelando a operação de *zoom*.

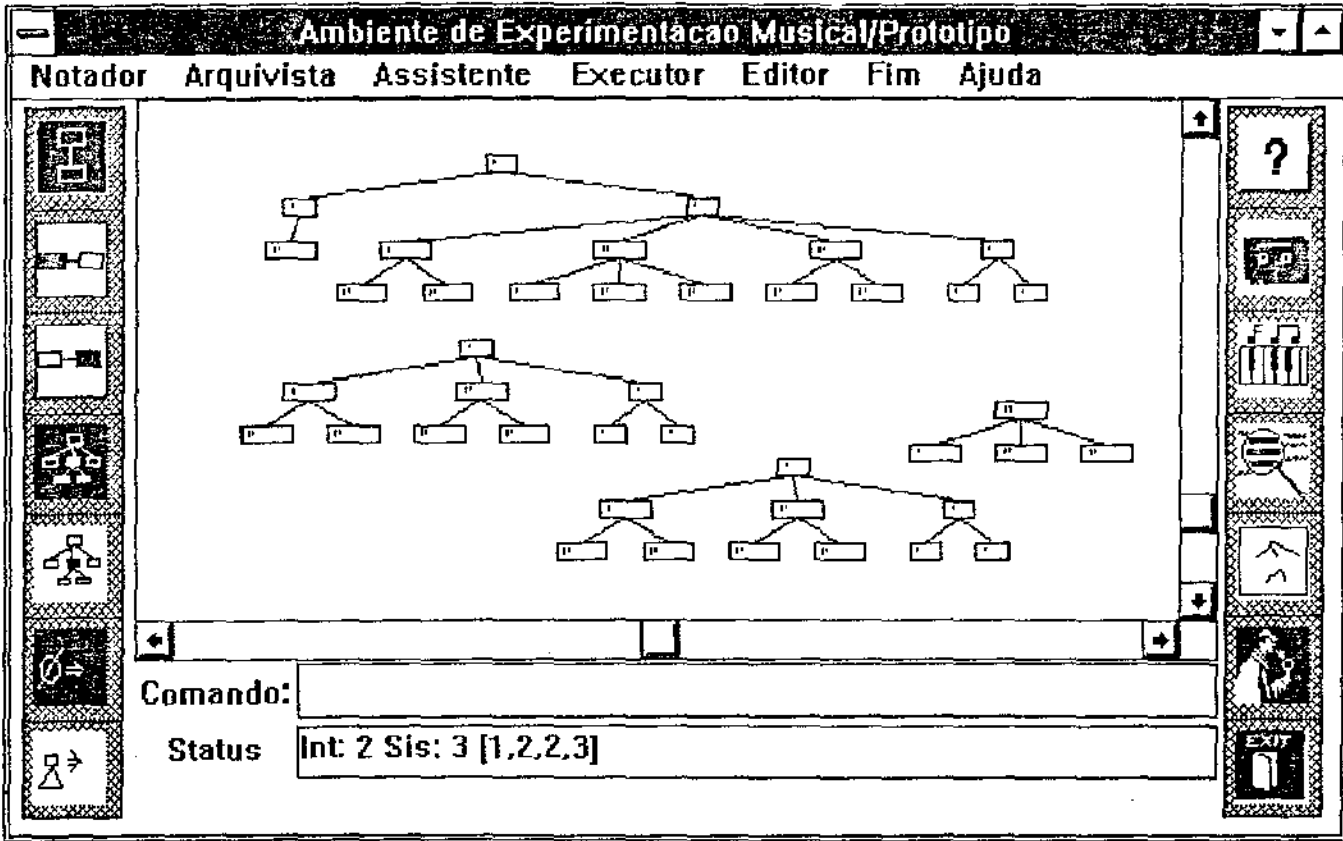


Figura 4.8: Um instante de interação com o sistema interativo

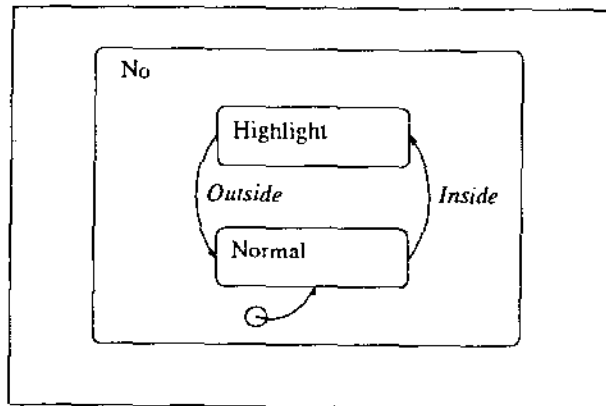


Figura 4.9: Uma simples especificação.

um lado tem-se um usuário que domina suas atividades e busca meios mais poderosos de realizar o seu trabalho. Por outro lado, o principiante preocupa-se primeiro com o uso da ferramenta que lhe é apresentada, para só então buscar o caminho que o especialista procura. Pensando neste aspecto, existem os vários estilos de diálogos citados anteriormente.

Pode-se fazer uma extensa descrição do protótipo em desenvolvimento, contudo, não seria um acréscimo necessário aos objetivos deste trabalho. A descrição limitou-se a comentar alguns cenários apenas. Está-se interessado no comportamento (controle de diálogo), que recebeu a maior parte da atenção deste trabalho.

#### 4.4 Visão geral do desenvolvimento da interface

O sistema especificado em [Bac] encontra-se na fase de projeto. A interface também se encontra nesta fase. A implementação realizada neste trabalho corresponde, em verdade, a um protótipo. Em §3.5.7 viu-se várias fases do ciclo de vida de um sistema e a importância dos protótipos para a avaliação do desenvolvimento de uma interface.

O projeto da interface, portanto, ainda está evoluindo. Constantemente são feitas alterações que são automaticamente repassadas para o protótipo. Esta pode não ser uma situação ideal de desenvolvimento, já que a interface (protótipo) encontra-se com mais de 7000 linhas! Infelizmente não se tem um construtor de protótipos disponível e eles são indispensáveis. No início do desenvolvimento não se tinha uma idéia definitiva de como seria a interface. Hoje tem-se uma idéia mais refinada que evolui paulatinamente a medida que novas funcionalidades são acrescentadas e em decorrência da própria interação com o protótipo.

Abaixo o desenvolvimento da interface é visto em duas etapas: projeto e implementação. Para cada uma delas são tecidos comentários relacionados ao desenvolvimento em execução.

#### 4.5 Projeto

Viu-se (§3.5.7) que o projeto compreende duas fases: a definição de requisitos e a fase de especificação. As seções abaixo, nesta ordem, refere-se sucintamente a estas atividades.

Conforme o ciclo de desenvolvimento estrela, figura 3.8, cada etapa pode influenciar as demais, bem como ocorrer em qualquer ordem. Apesar disto, abaixo segue uma ordem, o que não necessariamente reflete o processo de desenvolvimento. Ao se comentar a figura 1.2 (pág. 9) também foram feitas observações similares.

##### Definição de requisitos

Este estágio compreende várias atividades. Elas estão descritas em [Bac]. Está-se interessado na descrição do controle de diálogo, que é descrito na próxima fase baseado nos produtos

desta.

### Especificação

Uma das tarefas desta etapa é a definição de objetos de interação (§3.4). Eles representam as operações disponíveis para os usuários identificados no estágio anterior. Esta etapa inclui a definição de serviços gerais para permitir a realização de uma tarefa e a decisão de quem conduz a interação (usuário ou sistema). Os produtos deste estágio foram estabelecidos em [Bac]. Dos objetos de interação interessa a parte sintática e como se relacionam entre si. Embora em [Bac] seja apresentada uma especificação útil para a descrição de uma linguagem de comandos similar à tabela que descreve a funcionalidade do módulo **EDITOR** (pág. 56), muitos detalhes não são citados para outros estilos de interação, especialmente manipulação direta.

Este último estágio define a interface com a qual o usuário irá interagir. Neste ponto é interessante ressaltar os vários níveis de informações necessários para representar uma interface. A tabela supradita (tabela 3.1) exalta a sintaxe! Nota-se que para uso de manipulação direta o comando **crie nó** é disparado simplesmente clicando-se o botão esquerdo do mouse sobre região não ocupada por outro nó. Estes detalhes fornecidos por último referem-se a parte léxica da interface. A semântica equivale, neste instante, a disparar operação que cria internamente um nó.

### Construção de um protótipo

Embora este trabalho inclua a implementação de uma interface, tem-se, em verdade, a construção de um protótipo. Convém discernir claramente a implementação (parte deste trabalho, ou seja, um protótipo) com a fase de implementação posterior ao projeto de uma interface, comentada na seção seguinte. O projeto de uma interface é uma atividade iterativa, precisa de protótipos para validar o que o projetista está definindo. Só após a avaliação do projeto que prossegue a implementação, geralmente abandonando-se por completo o protótipo[MR92].

A implementação do protótipo prossegue (ainda em andamento) isolada do sistema. Ambos ainda correspondem a apenas um subconjunto que precisa ser expandido para se tornar útil. Este subconjunto, contudo, foi suficiente para os objetivos deste trabalho.

Atualmente tem-se mais de 7000 linhas de código dedicadas exclusivamente à interface. Estão distribuídas em vários arquivos em C++ e baseadas no paradigma de objetos, muito explorado no âmbito de interfaces (§3.8). A arquitetura de implementação segue o modelo de Seeheim. Nada impede, contudo, que a taxonomia definida em §3.4 seja utilizada para facilidade de identificação das funções desempenhadas por uma interface, como feito neste texto.

Usou-se o algoritmo descrito em [Moe90] para fornecer o leiaute de uma árvore. A árvore

vista na figura 4.9 foi obtida após fornecidas informações topológicas para o algoritmo que retorna posições geométricas para cada um dos nós.

A apresentação de informação é uma importante função de um UIMS. No entanto, no presente desenvolvimento não foi utilizado nenhum UIMS. Uma ferramenta de apresentação de informação que mantém consistência entre a apresentação e a informação que é representada é descrita por Kamada[KK91]. Não houve o apoio de nenhuma ferramenta deste tipo, o que tornou-se o “gargalo” do desenvolvimento. Observou-se que as alterações no diálogo eram facilmente transferidas para os Estadogramas que eram implementados de forma trivial. Mudanças na apresentação, contudo, sem o apoio de ferramentas mostrou-se um processo lento, complexo e que conduz a alterações *ad hoc*. Isto, sem dúvida, foi um fator inibidor de mudanças, o que jamais pode ocorrer no desenvolvimento de interfaces.

A maior dificuldade na construção do protótipo está sendo a camada dos objetos de interação (§3.4). Esta camada é parcialmente fornecida por *toolkits* (pág. 62) e encapsula muitos detalhes dos serviços fornecidos pelos sistemas de janelas. É importante observar que esta camada diz respeito a aspectos léxicos de uma interface. Está-se interessado na camada “controlador de diálogo” também visto em §3.4.

Esta dificuldade é oriunda parcialmente de uma deficiência da notação empregada: não há como registrar aspectos léxicos (que também fazem parte do diálogo). Tornar esta notação propícia à especificação de diálogo (não somente controle) implica em ampliar os recursos para permitir o registro de aspectos léxicos, como é comum em outras linguagens[Jac83, Was85].

## 4.6 Implementação

Na seção anterior discorreu-se sobre o projeto de interfaces. Esta seção descreve como é vista a implementação neste desenvolvimento.

No desenvolvimento realizado há uma especificação de um protocolo de comunicação através do qual a interface comunica-se com o restante do sistema e vice-versa. A especificação deste protocolo envolve não só o projetista da interface quanto o da aplicação. A partir do ponto que este protocolo “estabilizar-se” a implementação pode-se iniciar definitivamente. Não é necessariamente preciso, portanto, esperar todo o fim do projeto para que a implementação tenha início.

O que se tem para implementar definitivamente se se ainda está na fase de projeto? Novamente a figura 3.8 é útil. Nela inexistente uma ordem entre as várias atividades, que podem ser conduzidas das mais variadas formas possíveis.

A construção do protótipo visa assegurar um mínimo de qualidade do projeto. Não necessariamente todo o código deste protótipo será descartado. Por exemplo, o algoritmo para leiaute de árvore seguramente será utilizado na versão definitiva. Neste sentido, a interação hoje com o protótipo serve de teste da implementação deste algoritmo. Isto ocorre

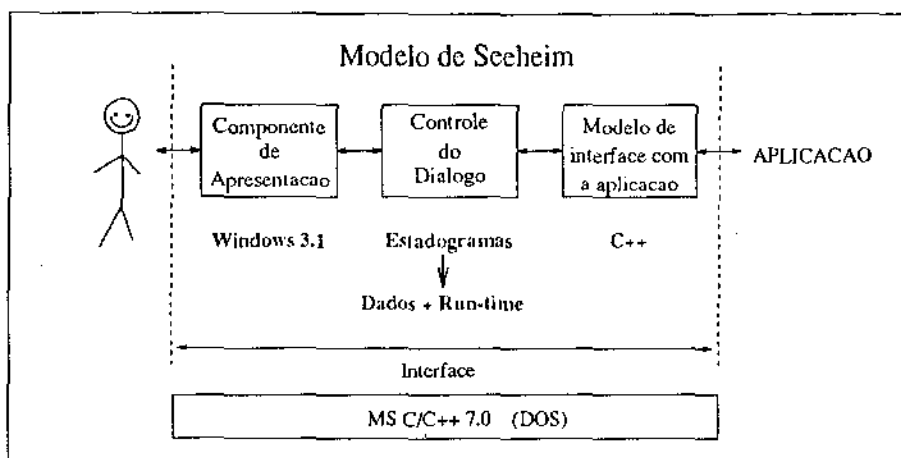


Figura 4.10: Ferramentas utilizadas.

de forma similar com relação a outros aspectos, sem inviabilizar mudanças e testes necessários sobre o protótipo.

## 4.7 Ambiente de Desenvolvimento

O ambiente de implementação sofreu mudanças (de software) durante o processo de síntese da interface. O ambiente *Windows 3.0* foi substituído pela versão mais recente (3.1). O compilador Zortech 3.0 foi preterido e o Microsoft C/C++ utilizado. O ambiente da Microsoft permite opções para gerar código com checagem forte de tipos (*strict type checking*), o que o torna mais rigoroso e menos propenso a difíceis erros que o da Zortech. Abaixo segue a descrição do ambiente final, ou seja, aquele em que os protótipos da interface e do sistema encontram-se implementados atualmente.

O sistema operacional é o MS-DOS V5.0 sobre o qual tem-se o *Windows V3.1*. O hardware compreende um IBM-PC/AT (386SX) com 8Mb de RAM.

A construção da interface em *Windows* contou com o SDK 3.1 (*Software Development Kit*), um depurador e várias ferramentas para programação em C++. O paradigma de objetos tem sido muito explorado no âmbito de interfaces (§3.8) o que torna C++ uma promissora linguagem, garantida pelo uso de C (muito utilizada [MR92]).

Para implementação dos Estadogramas foi utilizado a linguagem LEG e seu tradutor[Fil91b], contudo, só no início até a detecção de algumas dificuldades (comentadas no capítulo 2 e 5).

A figura 4.10 permite identificar ferramentas de suporte utilizadas e como elas se complementam.



Resumo:

### Software

- MS-DOS V5.0
- *Windows V3.1*
- SDK V3.1
- Microsoft C/C++ 7.0
- *Run-time* para núcleos reativos

### Hardware

- IBM-PC/AT 386SX
- 8Mb de RAM.
- Mouse
- Vídeo VGA (em cores)
- Placa MIDI
- Teclado (sintetizador)

O apêndice A fornece uma visão geral da construção de aplicações *Windows*.

## 4.8 Resumo

O capítulo precedente forneceu informações e conhecimentos necessários ao desenvolvimento proposto de uma interface. Ainda serviu de fundamento para questionar os Estadogramas. Este capítulo descreveu resumidamente um sistema para o qual uma interface está em desenvolvimento e seu estágio de desenvolvimento.

Ainda tratou-se da interface desenvolvida e questões correlatas. Foram fornecidas as etapas de desenvolvimento da interface, ainda em andamento, na fase de projeto com a construção de um protótipo. Alguns cenários da interação com o protótipo foram descritos com os Estadogramas equivalentes que os controlam.

Nada impede que o protótipo atualmente obtido venha a ser considerado a interface desejada ou conter parte dela. Protótipos podem incorporar paulatinamente características definitivas até que se tenha o produto acabado. Enquanto a fase de projeto não termina o desenvolvimento do protótipo prossegue. A construção deste protótipo equivale à implementação citada anteriormente como parte do trabalho.

Infelizmente a indisponibilidade de ferramentas para derivar o protótipo torna mudanças um processo lento. A maior parte do tempo atualmente empregado encontra-se no desenvolvimento de aspectos léxicos deste protótipo, já que a semântica não representa o foco deste trabalho. O aspecto sintático é analisado no capítulo seguinte. Pode-se adiantar, contudo, que este último aspecto não ofereceu resistência às alterações devido a existência de um *run-time* para realizar o comportamento de Estadogramas.

As sugestões feitas no capítulo seguinte foram idealizadas para simplificar ainda mais e aumentar os recursos da notação empregada. No capítulo seguinte ainda são comentadas algumas questões referentes à implementação. No capítulo 2 os Estadogramas foram descritos com algumas mudanças propostas e, por conseguinte, antecipado uma parte dos resultados da análise dos Estadogramas.

A dificuldade com os aspectos léxicos sugere considerações futuras. Por exemplo, seguindo a arquitetura descrita em §3.4, pode-se identificar um *toolkit* para a camada de objetos de interação e mecanismos para relacionar a notação dos Estadogramas às construções disponíveis no *toolkit*. A figura 4.11 exemplifica esta situação.

A inexistência do protocolo (figura 4.11) para uso com a notação dos Estadogramas faz com que o programador se preocupe em gerar os eventos lógicos para uso dos Estadogramas bem como todos os aspectos léxicos da interface. Isto retira dos Estadogramas um campo de aplicação fértil; construção rápida de protótipos, visto que seu uso seria mais adequado à implementação, já que não fornece recursos suficientes para tratar aspectos léxicos. A transferência de controle em pontos e situações particulares para rotinas predefinidas é um excelente recurso dos Estadogramas. Contudo, isto não é suficiente neste contexto.

A construção de especificações usando estadogramas não conta com “regras de outro.” Nenhuma referência consultada sequer faz alusão a princípios ou técnicas a serem mantidas

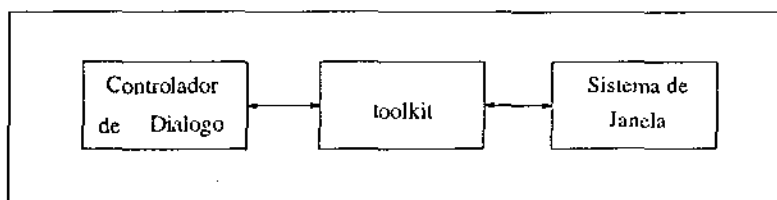


Figura 4.11: Fornecendo mais recursos à notação de Estadogramas.

em mente quando se constrói Estadogramas. Nesse trabalho o bom senso foi o único princípio. Naturalmente trabalhos deverão acumular experiências que poderão conduzir a princípios. Seguramente é desejável uma lista deles. Ela não irá resolver problemas hoje insolúveis, mas auxiliará quando o senso comum não indicar saída, além de servir de orientação.

## Capítulo 5

# Propondo alterações

*“Discussões acerca de especificações e linguagens de especificação seriam estéreis e fúteis sem a observação de suas aplicações a problemas reais e substanciais.”*

Gehani e McGettrick[GG86]

Este capítulo fornece várias propostas para uma notação candidata a representação de Estadogramas; alterações nos Estadogramas para suportar adequadamente a especificação de diálogo e restrições a serem satisfeitas na implementação destes diagramas para adequá-la ao software típico de uma interface.

As conclusões obtidas fornecem subsídios suficientes para outros trabalhos que venham implementar e validar as sugestões propostas. As conclusões resultam de um trabalho empírico sobre o desenvolvimento de uma interface real, onde os Estadogramas são utilizados para especificação do diálogo.

O capítulo anterior descreveu uma interface cujo diálogo foi considerado representativo e, portanto, suficiente para os objetivos deste trabalho. O emprego dos Estadogramas é realizado à luz de várias questões, comentadas no capítulo 3, acerca da atividade de desenvolvimento de interfaces.

---

### 5.1 Especificação formal

Enquanto a especificação informal apresenta problemas[Mey85], uma linguagem formal é de difícil compreensão e leitura conforme Melhart[MLJ89]. Melhart afirma que Estadogramas é um meio termo entre estes dois extremos. Não se deseja determinar se Estadograma é ou não uma linguagem formal. O objetivo é ressaltar a relevância da especificação formal e o trabalho de vários pesquisadores envolvidos com a semântica formal desta notação.

Em decorrência do custo elevado para corrigir erros tardiamente, métodos mais rigorosos (métodos formais) são desejáveis[Woo90]. [Abo91] é um trabalho que reconhece a necessidade de métodos formais e se concentra especificamente em interfaces. Convém ressaltar que a inexistência de métodos formais aplicados ao projeto de interfaces obriga a construção de protótipos para testes com os usuários.

Métodos formais não estão exclusivamente ligados à verificação de programas e podem ser utilizados para a especificação. Em outro extremo, a especificação informal é insuficiente e deve servir apenas de auxílio para gerar uma especificação formal[Mey85]. Idéias preconcebidas acerca da dificuldade de emprego, complexidade, aplicabilidade e outras referentes aos métodos formais são combatidas em [Hal90].

Não há uma definição precisa e largamente aceita para uma linguagem de especificação formal.<sup>1</sup> Contudo, do ponto de vista teórico, uma linguagem cuja semântica seja formalmente definida pode ser considerada uma linguagem de especificação formal. Não se tem a pretensão de dizer que Estadogramas representam uma linguagem de especificação formal, embora vários trabalhos (§2.6) preocupem-se com a sua semântica formal.

## 5.2 Algumas propostas

Esta seção discute várias questões relacionadas aos Estadogramas e ao âmbito de interfaces. O objetivo é identificar como esta notação e sua implementação se comporta diante de vários problemas relatados na literatura.

### 5.2.1 Modo

Foley et al.[FvDFH90, pág. 414] definem como o estado ou coleção de estados nos quais apenas um subconjunto de todas as possíveis tarefas de interação com o usuário podem ser realizadas. Em outras palavras, em modos distintos têm-se diferentes tarefas que podem ser realizadas. O uso de modo ainda permite que seja dado significado diferente para uma mesma entrada, conforme o modo corrente.

Interfaces que não apresentam o conceito de modo, ou aparentemente não apresentam, dificultam sua especificação utilizando estados e transições. Estes últimos elementos são essencialmente adequados quando modos estão presentes.

Conforme Myers[Mye89] uma das dificuldades com interfaces que usam manipulação direta é a inexistência de modos. Myers afirma que o usuário pode fornecer qualquer comando praticamente a qualquer momento. Isto tem implicações diretas no trabalho realizado. É importante ressaltar que Estadogramas modelam o comportamento como um conjunto de

---

<sup>1</sup>Transcrição da mensagem recebida da Prof<sup>a</sup>. Ana Lúcia C. Cavalcanti (UFPE-Recife). E-mail: alcc@di.ufpe.br.

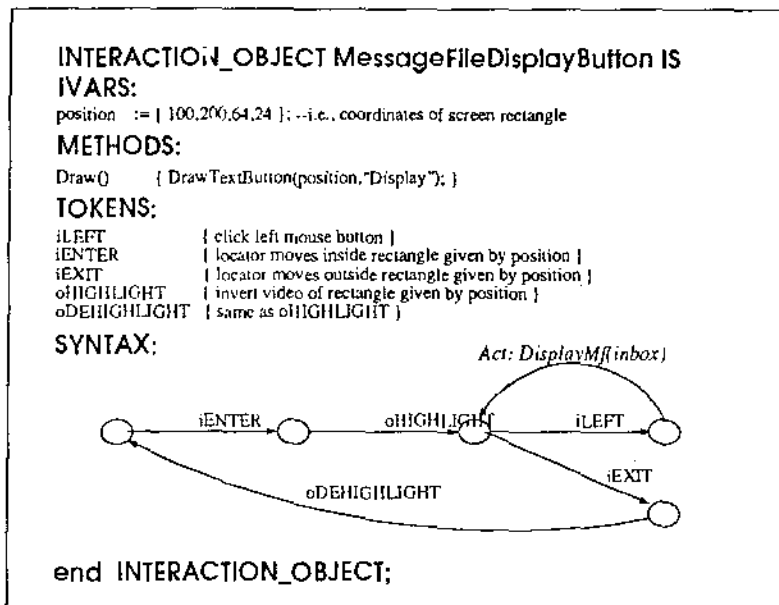


Figura 5.1: Objeto de interação [Jac86].

estados (equivalentes a modos) e transições entre eles. Surge um problema: como especificar tal interface se a noção de modos (estados) inexistente?

Jacob [Jac86] afirma que, apesar da aparência, manipulação direta é altamente modal! Jacob mostra a estrutura de co-rotinas<sup>2</sup> existente nestas interfaces, usa diagramas de transição de estados para representar o estado interno de cada uma delas, e propõe uma linguagem para descrevê-las. Cada co-rotina equivale a um objeto de interação. Um objeto de interação na linguagem definida por Jacob pode ser vista na figura 5.1.

Jacob sugere que cada *locus* de diálogo seja descrito como um objeto de interação que pode reter seu estado através do diagrama de estados. Um controlador fica encarregado de coletar todos os diagramas de estados dos objetos de interação e executá-los como uma coleção de co-rotinas. Neste ponto é-se inclinado a dizer que os objetos de interação nada mais são que estados concorrentes com *history*, em que a transição para os mesmos segue apenas até a borda do estado ancestral. Naturalmente um caso particular do comportamento dos Estadogramas. Entretanto, Jacob afirma que o usuário não vê uma interface como um grande diagrama de estados, mas uma coleção de muitos e semi-independentes objetos. A especificação proposta por Jacob pode ser vista como uma “casca” sobre a descrição de um estado em Estadogramas. Nota-se que a seção **TOKENS** (figura 5.1) nada mais é que

<sup>2</sup>Co-rotinas (*coroutine*) é uma variação de uma rotina comum ou sub-rotina. Quando uma rotina chama outra, a execução tem início fixo no ponto de entrada. Ao finalizar a execução ocorre um retorno para o local de chamada. Co-rotinas não possuem retorno, mas transferência de controle para outras rotinas. Nesta transferência a execução não se inicia no ponto de entrada (como nas rotinas comuns), mas no último ponto executado na co-rotina chamada.

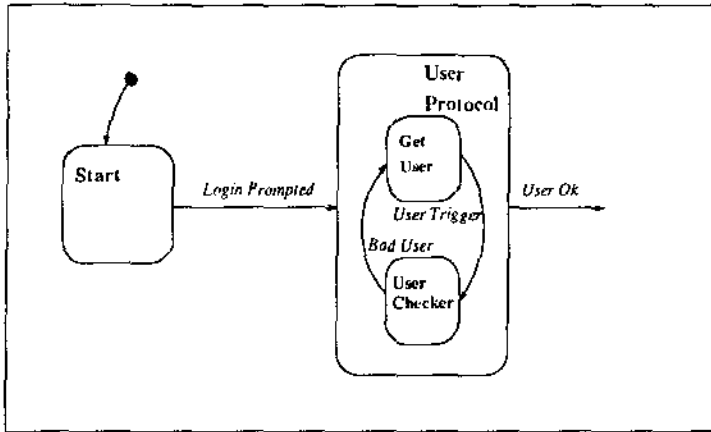


Figura 5.2: Usando estado para modelar processo.

a relação entre eventos lógicos e físicos. Muitas linguagens para especificação de diálogo permitem este mapeamento explícito.

Neste ponto é interessante fazer uma comparação informal entre Estadogramas e a linguagem descrita por Jacob. Com o uso de Estadogramas têm-se disponíveis recursos apenas para a descrição da sintaxe. Os recursos, contudo, incluem concorrência que não necessariamente precisa ser vista como diagramas de transição de estados que operam independentes uns dos outros. Uma deficiência dos Estadogramas fica clara quanto ao aspectos léxicos (apresentação) de uma interface. Não há como fornecer um mapeamento entre eventos físicos e lógicos, nem como exibir um simples caractere. Vimos anteriormente, contudo, que aspectos léxicos podem coexistir isoladamente da sintaxe como em [Sin89]. As observações de Jacob são desfavoráveis aos Estadogramas quanto a descrição completa, em um único diagrama, do diálogo do sistema interativo. Isto também pode ser resolvido se Estadogramas forem vistos como uma coleção de estados ortogonais (semelhante a visão de diagramas de transição de estados por Jacob). Nesta última hipótese ainda restaria um fator favorável aos Estadogramas: a comunicação entre estados. Um outro fator seria a sua estrutura hierárquica.

Outra sutileza existe quanto ao preciso conceito de estado modelado em Estadograma. Por exemplo, o estado *User Protocol* (figura 5.2, extraída de [Fil91a, pág. 39]) contém um subestado *User Checker*. Este subestado, no entanto, modela um processo: verificar a existência do usuário. O evento *Bad User* pode ser visto naturalmente como o retorno de uma função *UserChecker()*. Neste caso um estado foi utilizado para modelar um processo invisível ao usuário. A figura 5.3 é uma modelagem alternativa e mais substancial que a anterior para o usuário, pois reflete o que ele percebe. Conclui-se daí que especificações como a da figura 5.2 devem ser evitadas.

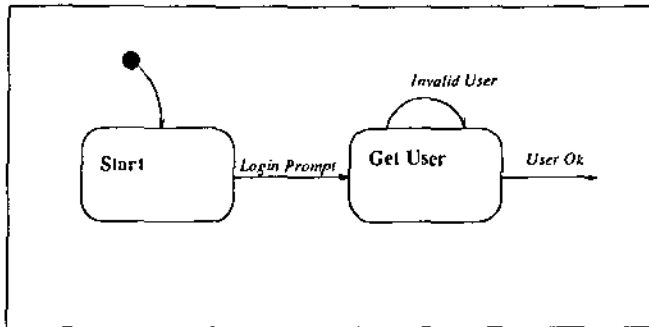


Figura 5.3: Usando estado para modelar o que o usuário percebe.

### 5.2.2 Callbacks functions

Sistemas de janela como *X Window* e *Windows* permitem fazer uso de *callbacks functions*. Nestes casos, sempre que determinado evento ocorre automaticamente tais sistemas chamam a *callback function* apropriada.

O código gerado pelo Tradutor exige uso de *callbacks*, o que muitas vezes obriga a criação de funções com a única finalidade de atender esta exigência. Evitar as funções e inserir código C em código LEG (conforme trecho abaixo) não é uma solução razoável, pois obriga tornar globais variáveis e código de outros módulos.

Abaixo segue a especificação de um simples Estadograma escrita em LEG. Logo a seguir tem-se o código gerado a partir desta especificação pelo Tradutor.

```

main blob SimpleExemplo
{
  blob EstadoInicial
  {
    on_entry : [ printf("\nEstado Inicial ativo.");
                err = Inicializacao(); /* Inicializa sistema */
                ];
    transition { on_event(InicializacaoOK) to Normal }
  }
  blob Normal { /* ... */ }
  blob Fim
  {
    on_exit : [ printf("\nExitAreas"); ];
    on_entry : [ err = Finalizacao(); ];
    transition {
      on_event(FinalizacaoOK) to Inativo
    }
  }
}

```



```
blob Inativo { /* ... */
}
```

Parte do código gerado da especificação anterior:

```
#include "output.h"

void OnEntry(int no)
{
  switch( no )
  {
    case EstadoInicial : printf("\nEstado Inicial ativo.");
                        err = Inicializacao();

    break;
    case Fim : err = Finalizacao();
    break;
  }
}
```

A função `OnEntry` é sempre chamada pelo *run-time* quando se entra em algum estado. O argumento `no` contém o identificador do estado que está sendo ativado. Se houver alguma ação a ser executada na entrada do nó no ela será identificada pelo `switch` e executada. O Tradutor blob gera uma função semelhante para a saída (`OnExit`) e outra para ações associadas às transições.

Para evitar as *callbacks* uma solução imediata seria permitir que os módulos responsáveis pela ação a ser executada realize esta operação. Por exemplo, na especificação LEG acima vê-se ações de competência do módulo de apresentação (aspectos léxicos, `printf`) e ações pertinentes a aplicação (semântica, `Inicializacao` e `Finalizacao`). A solução compreenderia uma forma de comunicação entre estes módulos de forma que as *callbacks* fossem evitadas.

A proposta de um UIMS em [Sin89] adota esta abordagem. No lugar das *callbacks* estariam mensagens a serem enviadas ao componente apropriado. O controlador de diálogo não chama diretamente rotinas da aplicação ou responsáveis pelo tratamento léxico da interface. Em vez disso coloca em uma fila *tokens* que identificam para estes componentes o que eles devem fazer. Fica a cargo dos componentes responderem adequadamente a estes *tokens*. Isto ainda permite um relacionamento assíncrono entre os componentes de um sistema interativo, o que muitas vezes é desejável. A figura 5.4 exemplifica esta situação.

A figura 5.4 originalmente ([Sin89]) não possui passagem de *tokens* da aplicação para o controlador de diálogo, ou seja, o arco ligando a aplicação à fila de *tokens* para o controlador de diálogo não está disponível. Uma consequência desta limitação é que o controle nunca reside na aplicação. Há um controle misto no UIMS de Singh, pois pode requisitar

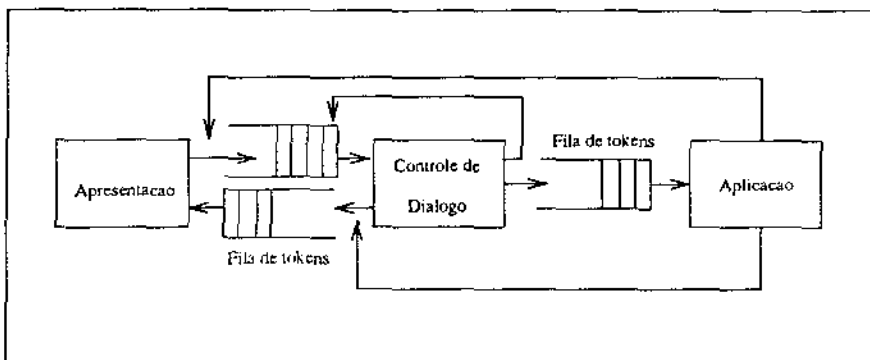


Figura 5.4: Comunicação entre componentes de sistemas interativos. Adaptado de [Sin89].

informações acionando diretamente o módulo de apresentação. Entretanto, não há como controlar o diálogo.

A comunicação entre os módulos, que podem operar concorrentemente, se faz através da passagem de *tokens* eliminando as *callbacks* e isolando ainda mais os componentes de um sistema interativo. Neste caso, é importante notar que não seria preciso sobrecarregar a notação para especificar o controle do diálogo com detalhes léxicos como feito em [Was85] e [Jac86], por exemplo. Uma outra notação poderia ser fornecida para responder adequadamente aos *tokens* léxicos recebidos e implementar a apresentação da interface.

A descrição do tratamento do evento *IN\_CHECK* na próxima seção exemplifica o mecanismo de comunicação do controlador do diálogo com a apresentação e a aplicação. A função *send\_token* coloca, na fila especificada, um argumento fornecido como mensagem.

### 5.2.3 Variáveis

Outro recurso que simplificaria a especificação de um controle de diálogo é o uso de variáveis (comuns nas linguagens de programação convencionais). Variáveis já são empregadas pelo modelo de eventos.

Até o momento utilizou-se estados apenas para representar determinada situação. Não se pode desprezar a utilidade de variáveis. Muitas vezes um modo é modelado mais simplesmente através de uma variável lógica (verdadeiro ou falso) do que com um estado ortogonal com dois subestados. Variáveis ainda são úteis para transferir informações do componente de apresentação para a aplicação e vice-versa. Por exemplo, conforme a arquitetura descrita na seção anterior, ao se selecionar uma opção de um menu o módulo de apresentação envia um *token* para o controle de diálogo com informações sobre a opção escolhida. Este *token* deve ser mapeado pelo controle de diálogo em outro compreendido pelo módulo da aplicação, juntamente com informações necessárias para a operação. Sem o uso de variáveis isto deve ser feito por um dos componentes, o que não reflete o caráter de mediador do controle de diálogo. O exemplo abaixo, extraído de [Sin89], exemplifica uma situação em que o uso de

variável foi útil para servir de intermediário entre a aplicação e a apresentação:

```
event IN_CHECK
{
    if (Change_Root_position.status == DEF) {
        values = (int*) calloc(1,sizeof(int));
        values[0] = Change_Root_position.value;
        send_token(APPLICATION,INPUT, Change_Root,values);
        send_token(PRESENTATION,INITIAL,"cmenu","-1");
        cmd_Change_root.status = OFF;
        destroy_instance(self_id);
    }
}
```

Embora trate-se de trecho de um tratador de eventos (exemplo acima), ele reflete operações que devem ser realizadas independentemente da notação ou modelo de diálogo utilizado. Se não existem variáveis na notação dos Estadogramas, então algum módulo terá que ser encarregado de realizar o mapeamento exemplificado acima. Esta última opção, contudo, não é razoável pelos propósitos que se está interessado em registrar com a notação dos Estadogramas.

#### 5.2.4 Depuração

Em seção anterior comentou-se o uso de *callbacks*. Outro inconveniente desta prática diz respeito a depuração do código gerado. Nesta mesma seção foi visto um trecho de código gerado pelo Tradutor. Observa-se que a depuração do código gerado inevitavelmente mostrará para o programador código que não é seu, dificultando esta atividade.

Uma implementação que suporta o controle de diálogo como na figura 5.4 facilita a depuração, pois permite localizar com mais facilidade a origem de um erro.

#### 5.2.5 Reutilização de comportamentos

Embora a especificação em Estadogramas possa seguir *bottom-up* ou *top-down* com a especificação paralela de vários estados e por pessoas distintas, é preciso que estes recursos não sejam exclusivos de uma especificação particular.

O tratamento de forma monolítica de um estadograma impede que existam bibliotecas de comportamentos que possam ser reutilizados no desenvolvimento de várias especificações.

Usando-se uma linguagem de programação convencional pode-se criar rotinas e colocá-las em uma biblioteca para uso posterior, sem a necessidade de ter acesso ao código destas rotinas. Da mesma forma é preciso estabelecer uma forma de reutilizar comportamentos sem

a necessidade de termos acesso a sua especificação. Estes comportamentos “reutilizáveis” corresponderiam à sintaxe de objetos de interação comumente encontrados em sistemas interativos. Por exemplo, poder-se-ia ter diálogos para procurar por arquivos, alterar parâmetros de impressão, selecionar cores e assim por diante. Todos estes comportamentos com objetivos bem definidos, o que permitiria soluções aplicáveis em várias situações.

### 5.2.6 Alteração dinâmica do comportamento

O desenvolvimento interativo de protótipos é amplamente perseguido por sistemas de apoio à construção de interfaces. Nestes sistemas, o comportamento é alterado dinamicamente e imediatamente o sistema reflete as alterações. Usando-se Estadogramas é preciso estabelecer uma semântica bem definida para alterações realizadas. Por exemplo, ao simular uma especificação o que ocorre se removermos um estado corrente? Existem outras situações que possuem variadas interpretações. Estes casos devem ser bem estabelecidos.

Mas esta não é a única aplicação da mudança dinâmica de uma especificação. Os tratadores de eventos [Gre86] podem ser criados e destruídos dinamicamente. Várias instâncias de um mesmo tratador podem estar ativas simultaneamente. Para os Estadogramas ter-se-ia que permitir a existência de várias “instâncias” de um estado.

Novamente, a semântica de tal instância deve ser estabelecida de forma clara, bem com os casos conflitantes oriundas da sua existência. Este acréscimo fará, provavelmente, com que cada estado tenha um identificador único. O identificador permitiria informar qual instância de um estado irá receber um evento, por exemplo.

Aparentemente tal recurso pode ser sem utilidade. Contudo, qualquer editor que permite a edição simultânea de vários arquivos mostra a necessidade destes recursos. É estranho imaginar que se tenha a especificação de um editor para cada arquivo em edição e não uma única especificação com múltiplas instâncias. Um outro recurso a ser adicionado nesta situação é a comunicação entre estas instâncias, para permitir a realização de atividades como *cut* e *paste* entre os arquivos em edição.

### 5.2.7 Undo

É comum sistemas fornecerem uma maneira do usuário reparar uma operação disparada por engano ou erro. Isto estimula o uso do sistema! O usuário não se sente inibido, pois as operações são reversíveis.

Até o momento ações do usuário quase sempre causam transições entre estados. Como seria, portanto, o inverso de uma transição? Percorrê-la em sentido contrário? Simplesmente voltar à configuração anterior?

A resposta dos Estadogramas sempre é na forma de ações. Portanto, esta é a única maneira como os Estadogramas comunicam-se com o exterior ou o modificam. Parece razoável

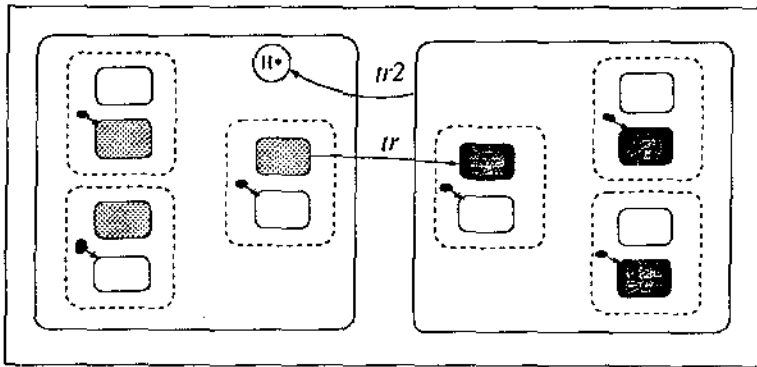


Figura 5.5: Uma proposta para *undo* ( $\text{undo}(tr) = tr2 + \text{ações}$ ).

que após um *undo* os Estadogramas retornem a última configuração em que estiveram. A dúvida surge quanto às ações a serem executadas neste “retorno.”

Viu-se no capítulo 2 que uma transição é o resultado de se percorrer uma árvore que representa uma hierarquia de um Estadograma, gerando ações sempre que se visita ou abandona um nó. Se o efeito for percorrer a árvore em sentido contrário então trata-se de um caso particular de transição, mas isto pode não ser desejado. Nem sempre a ação executada pelo fato de se entrar em um estado tem o efeito oposto da saída.

Um *undo* geralmente envolve ações específicas a serem realizadas tanto pela aplicação quanto pelo módulo de apresentação. Portanto, uma maneira natural de se observar esta situação é permitir que sejam especificadas ações de *undo* para cada transição. Assim, sempre que um *undo* ocorrer, as ações necessárias serão executadas.

Em cinza (figura 5.5) tem-se a configuração dos Estadogramas antes da transição *tr*. Em preto tem-se a próxima configuração como conseqüência da transição *tr*. Após a entrada em todos os estados em preto o sistema se estabiliza. Neste instante, se ocorrer um *undo* a próxima configuração será a cinza. Esta configuração é a mesma configuração obtida pela transição *tr2* devido ao *history H\**, exceto se houver um cancelamento de *history* nos níveis inferiores. Viu-se que o problema é estabelecer quais as ações que devem ocorrer. Conforme a proposta, os estados em cinza estarão ativos ao fim do *undo* e as ações especificadas para esta situação terão sido executadas.

*Undo* ocorre com frequência nos sistemas interativos e os Estadogramas devem possuir uma construção para facilitar esta especificação. Isto pode ser afirmado baseando-se em propriedades desejáveis em uma linguagem de especificação.

### 5.2.8 Escopo de eventos

Conforme exemplificado no capítulo 2, a figura 2.14 (pág. 32) exemplifica uma transição que jamais ocorrerá. Trata-se da transição mais interna rotulada com o evento *a*. Na figura 5.6,

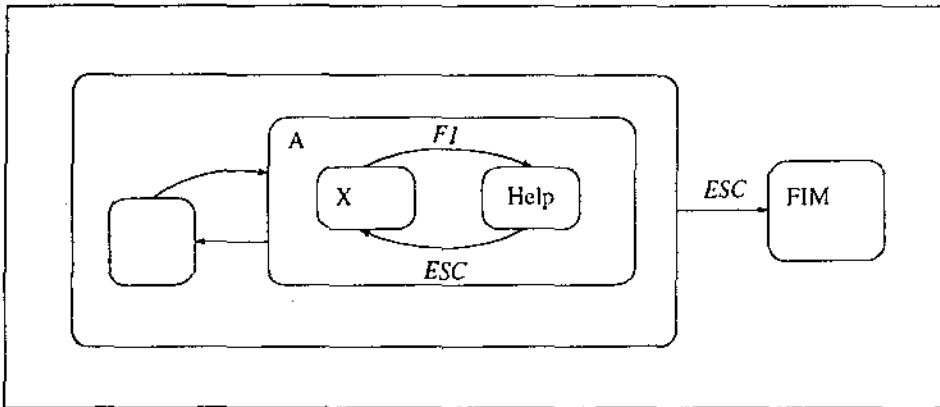


Figura 5.6: Definindo contexto para eventos.

contudo, é apresentado um caso onde esta situação é indesejável.

Em determinado instante (estado **X**) o usuário requisita informações de ajuda (gera o evento *F1* possivelmente através da tecla *F1*) causando a transição para o estado **Help**. Geralmente isto equivale a uma caixa de diálogo que exibe informações sobre determinado comando, por exemplo. Para prosseguir com a operação do sistema é necessário que o usuário saia do estado de ajuda. Na especificação da figura 5.6 isto é feito pressionando-se *ESC*. Este evento gerado, contudo, jamais provocará uma saída do estado **Help** devido a transição de maior prioridade também rotulada com o evento *ESC* (conforme visto na pág. 32).

Esta situação é indesejável, já que para garantir a consistência (§3.5.2) deve-se ter um tratamento homogêneo para atividades afins, neste caso é fim da ajuda e finalização do sistema. Da mesma forma espera-se que toda e qualquer caixa de diálogo possa ser removida com a tecla *ESC*, bem como menus e assim por diante.

Para eliminar esta situação pode ser estabelecido um escopo (estado) para cada evento. Por exemplo, pode-se estabelecer que o estado **Help** (figura 5.6) é o escopo do evento *ESC*. Sempre que **Help** for o estado ativo o evento *ESC* pode afetar apenas a transição que sai deste estado. Isto não afeta subestados de **Help**. O estado **A**, por exemplo, não deve ser o escopo deste evento porque evitaria finalizar o sistema quando o estado **X** estiver ativo, o que não é a intenção.

Outras situações interessantes podem ser obtidas com este conceito. Por exemplo, pode-se definir estados cujo único objetivo é restringir o efeito de alguns eventos, i.e., inibi-los. A figura 5.7 mostra outra situação interessante. Ainda inclui a proposta aqui apresentada para representar que o escopo de um evento, ou seja, sempre que evento estiver entre chaves, este evento terá como escopo o estado da origem da aresta rotulada com este evento.

Após a entrada no estado **A** o primeiro evento *e* causa apenas a transição no estado ortogonal da esquerda. Após esta transição um outro evento (*e*) causa a transição no componente da direita.

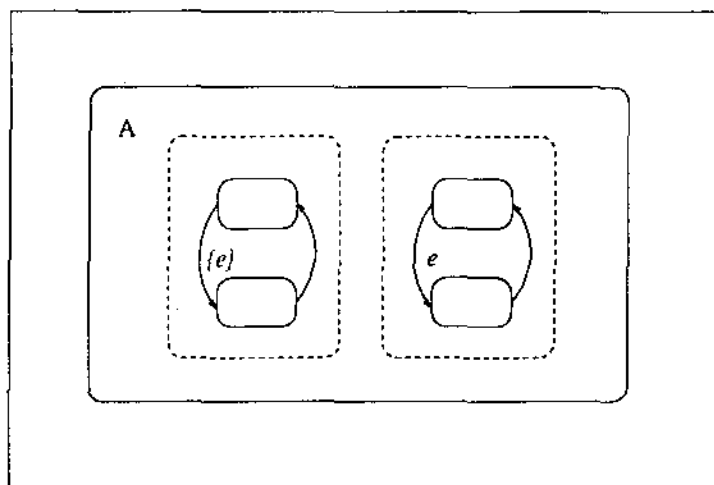


Figura 5.7: Escopo de eventos.

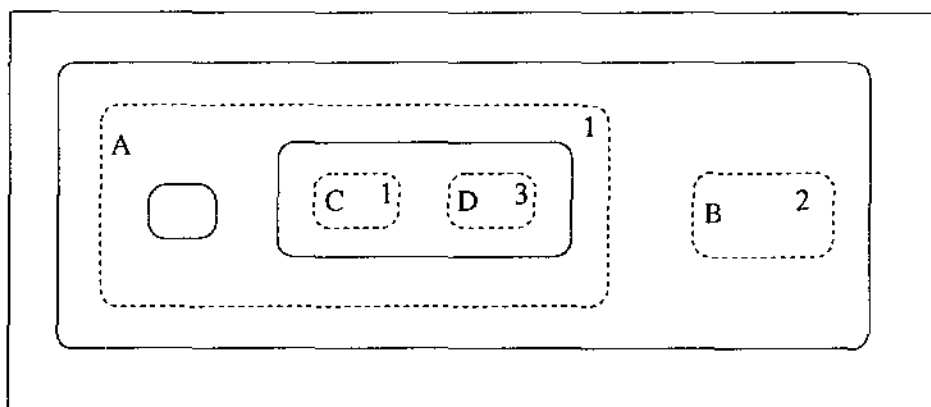


Figura 5.8: Concorrência em níveis distintos.

### 5.2.9 Estabelecendo prioridades

Na figura 2.13 (pág. 31) vê-se como o determinismo é realizado em componentes concorrentes. No entanto, a ordem imposta na consideração dos estados naquela figura era arbitrária e fixa. Surgem situações, entretanto, em que não é possível usufruir adequadamente da concorrência disponível se inexistir um recurso mais flexível. A figura 5.8 exemplifica um caso. A dificuldade surge quando tem-se concorrência em níveis hierárquicos distintos.

Os estados A e B são estados concorrentes. Em determinado instante pode-se ter ativos os estados C, D e B. Para tornar determinístico as ações em estados concorrentes viu-se na figura 2.13 como eventos são tratados em estados ortogonais. Aquela solução proposta, contudo, não contempla as sutilezas oriundas de uma hierarquia. Por exemplo, pode-se considerar primeiro o estado A e só então o estado B ou vice-versa. Não há contudo, algo

como **D**, **B** e **C**, pois intercala estados concorrentes em níveis hierárquicos distintos. A solução é permitir que esta ordem possa ser especificada.

A numeração dos estados na figura 5.8 exemplifica esta situação. No caso de ocorrências que afetam os estados **A** e **B** são tratadas primeiro as que se referem ao estado **A** e só então o estado **B**. Caso **C**, **D** e **B** sejam os estados ativos a ordem é **C**, **B** e **D**. Se houver colisão nas prioridades pode-se assumir uma ordem arbitrária.

## 5.3 Uma notação em forma de texto para Estadogramas

Usar terminais gráficos para editar/simular e analisar especificações de Estadogramas em forma de diagramas são, seguramente, preferíveis a trabalhar com descrições em forma de texto. Estas últimas são mais suscetíveis a erros e tornam a interpretação mais difícil. Entretanto, nem sempre se possui o hardware necessário.

Convém ressaltar que a representação gráfica de Estadogramas são valiosas para o uso humano, mas de pouca utilidade para um *run-time*. Um *run-time* necessita de uma árvore que representa a topologia e tabelas para as transições.

Estas observações tornam necessária a existência de outra linguagem que possa ser compreendida pelo *run-time* (ou que possa ser “compilada” gerando código e/ou tabelas para o *run-time*) e de fácil manuseio para uso humano. LEG tem estes objetivos. O texto abaixo, contudo, mostra que mudanças são necessárias para fazer desta linguagem outra mais expressiva e legível.

### 5.3.1 Transições

Na especificação de LEG transições ocorrem sempre entre dois estados. Por exemplo, o trecho abaixo descreve uma transição do estado **Origem** para o **Destino** quando o evento *e* ocorrer e **Origem** for o estado corrente:

```
blob Origem
{
  transition
  {
    on_event(e) to Destino
  }
}
```

Nota-se que esta forma de descrever uma transição possui inconvenientes, pois ao se observar a descrição de um estado, desconhece-se os estados que atingem este estado por alguma transição. Por exemplo, não se sabe observando a descrição em LEG acima em que



circunstâncias o estado **Origem** será ativado. Ao observar uma especificação ter-se-ia que procurar por sentenças `transition` em toda a descrição. Nesta busca seria identificado os estados que atingem o estado **Origem**.

Esta dificuldade é consequência de descrever uma transição a partir do seu estado origem. Se a descrição for do estado destino tem-se um problema simétrico. Descrever em ambos resulta nos inconvenientes de redundância de informações.

Uma alternativa a ser experimentada é descrever transições no “escopo” em que ocorrem. O escopo de uma transição é o ancestral comum mais próximo do estado origem e destino da transição. A descrição em LEG equivalente ao Estadograma da figura 5.9 seria:

```
blob Outermost
{
  blob A; /* A e B podem ser especificados em outro lugar */
  blob B; /* ou estarem disponíveis em uma biblioteca */
  transition
  {
    on_event(e1) from { O1,O2 } to { D1,D2 };
    on_event(e2) from C to B;
  }
}

blob F
{
  transition
  {
    on_event(e3) from D1 to D3
  }
}
```

Vê-se que as transições estão agrupadas conforme o escopo de cada uma. Nota-se que isto facilita mais a compreensão do que ocorre durante uma transição. Na forma anterior existe esta informação, mas de forma implícita. Observa-se que localizar transições torna-se tarefa relativamente simples, pois é suficiente identificar o escopo da transição e percorrer um subconjunto de todas as transições, no pior caso.

Na descrição original da LEG não há como especificar as transições que vão de um conjunto para outro de estados. A transição rotulada com o evento *e1* exemplifica este caso. As transições são apenas de um estado para outro.

A disponibilidade de transições como a rotulada pelo evento *e1* na figura 5.9 gera outra necessidade. Identificar a ordem em que o conjunto de estados será deixado (**O1,O2** ou **O2,O1**) e a ordem de entrada no estado **B** (**D1,D2** ou **D2,D1**). Novamente uma número

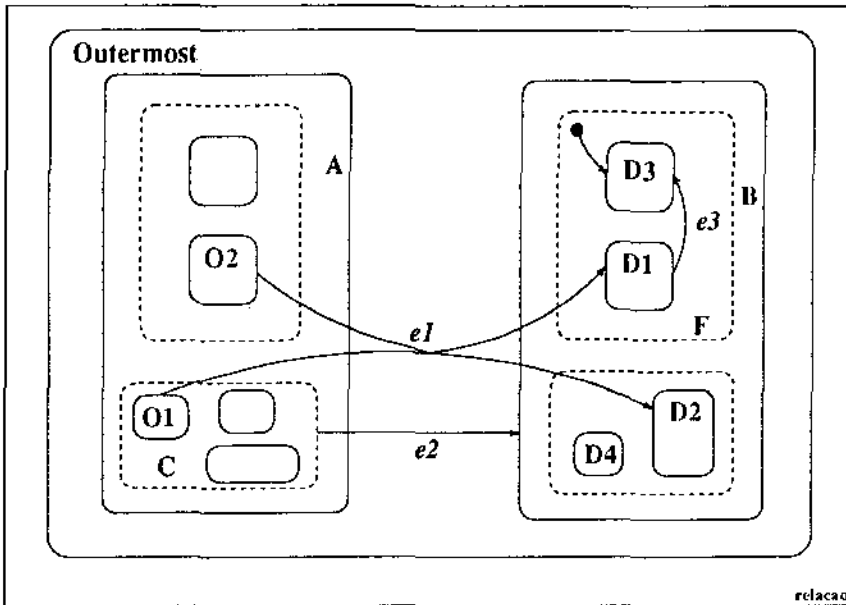


Figura 5.9: Transição envolvendo conjuntos de estados.

associado a cada transição pode ser um solução para que o comportamento possa ser descrito da forma tão flexível quanto desejado.

Poder-se-ia estar valorizando desnecessariamente a ordem em que estados concorrentes são tratados e a ordem em que transições ocorrem. Entretanto, para uma interface desconsiderar tal ordem pode conduzir à efeitos indesejáveis. Por exemplo, na interface em desenvolvimento, mesmo que seja utilizada manipulação direta para criar um nó (função do módulo **EDITOR**, tabela 3.1) a linha de comandos imediatamente reflete, na linguagem de comandos, o comando equivalente à ação do usuário usando o mouse. Parece mais razoável que este comando seja exibido, só então o nó é criado internamente (aplicação) e exibido (apresentação). Nada impede que esta característica do sistema seja modelada como estados concorrentes. Neste caso seria necessário ditar esta ordem. Ainda pode-se pensar em alterar dinamicamente esta ordem. Por exemplo, o usuário pode estabelecer outra ordem com custo mínimo.

### 5.3.2 Transição alternativa

A necessidade de especificar todas as interações possíveis usando transições entre estados constitui um dos inconvenientes dos diagramas de transição de estados. Esta necessidade existe se se desconsidera a existência de uma outra linguagem que possa complementar uma especificação. Em outras palavras, se toda a especificação do comportamento for realizada usando-se os diagramas de estados, então todos os possíveis caminhos devem ser previstos como transições entre estados; nada impede entretanto, que os diagramas representem ape-

nas parte da especificação, enquanto outra linguagem convencional (C, Pascal) completa a especificação. O protótipo até o momento desenvolvido utiliza esta abordagem, ou seja, parte do controle é realizado em C++ e parte em Estadogramas.

Uma forma alternativa de amenizar este problema é fornecer uma espécie de transição *default* que sempre é percorrida quando a entrada do usuário não gerar nenhuma outra transição. Este tipo de transição, portanto, distingue-se de outra transição *default* vista na página 26. A construção em LEG poderia ser:

```
blob C
{
  transition
  {
    automatic to A do Display("Try again"); /* lambda transition */
    default to B
  }
}
blob A
{
  transition
  {
    on_event(e) do action(); /* Escopo desta pseudo-transicao e o */
  }                               /* proprio estado                */
}
```

Observe a palavra *default* precedendo a sintaxe normal de uma transição. Graficamente pode-se ter transições tracejadas indicando esta característica. A figura 5.10 mostra os Estadogramas equivalentes. A especificação em LEG também exemplifica a pseudo-transição vista no capítulo 2.

A figura 5.10 exemplifica estas alterações. Observe que a transição *t1* é diferente de *t2*. *t1* é uma transição *default* enquanto *t2* indica uma transição- $\lambda$ . Sempre que o estado A estiver ativo e não possuir transição para tratar um evento gerado percorre-se a transição *t1* conduzindo ao estado B. O estado B possui uma transição- $\lambda$ , ou seja, sempre que atingido esta transição é percorrida. Isto retorna ao estado anterior após a mensagem “*Try Again.*”

## 5.4 Implementação e suas implicações

Viu-se anteriormente dificuldades inerentes a representação de interfaces e algumas pertinentes à implementação. Esta seção trata do preço de se empregar o conceito de independência de diálogo juntamente com manipulação direta.

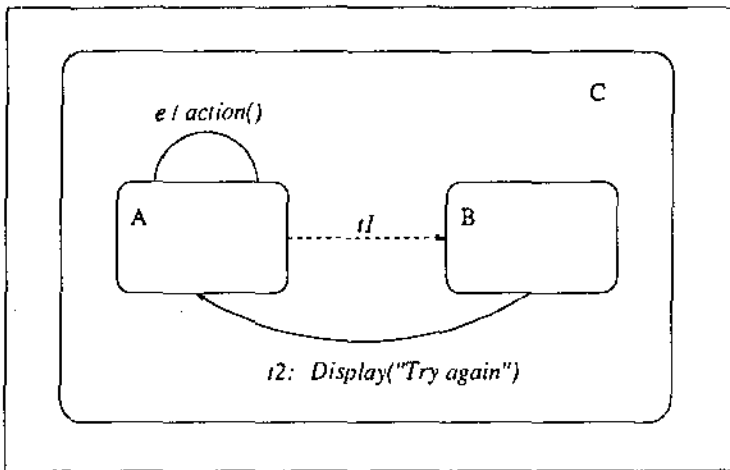


Figura 5.10: Transições: pseudo, *default* e  $\lambda$ .

#### 5.4.1 Manipulação direta

As dificuldades com a implementação de interface que usam manipulação direta (§3.2) são bem conhecidas[SV91, Mye89, HH89a].

Entre as dificuldades encontra-se o suporte do princípio de realimentação (§3.5.2). Interfaces que utilizam de manipulação direta permitem que a entrada possa ocorrer concorrentemente com a realimentação. Esta situação dificulta a implementação seqüencial. Um ambiente que suportasse concorrência entre a interface e a aplicação seria o ambiente ideal.

Em ambos os casos, contudo, surge uma nova dificuldade: *overhead* de comunicação entre a interface e o sistema. Por exemplo, quando o usuário trabalha com um gerenciador de arquivos sobre um sistema de janela, geralmente ele pode “arrastar” (*drag*) um arquivo e “soltá-lo” sobre um ícone. Se o ícone representar uma lata de lixo este arquivo é removido, se representar uma impressora será impresso e assim por diante. Entretanto, estas simples operações necessitam de muitas informações semânticas que não estão, ou melhor, não deveriam estar na interface. Se não estiverem na interface para manter os benefícios da desejada independência de diálogo, então estão na aplicação. Se estiverem na aplicação, então torna-se necessária a comunicação entre a interface e este componente para que tais informações possam ser obtidas e que o usuário tenha a realimentação adequada. Convém ressaltar que a realimentação para o usuário deveria ser imediata. Seria lamentável mover o mouse para só instantes após o cursor refletir a nova posição.

O estilo de manipulação direta é a origem dos problemas. Refletir imediatamente as alterações sobre os objetos manipulados exige tráfego intensivo de informações entre a interface e a aplicação. “Aproximar” estes componentes de um sistema interativo contraria o conceito de independência de diálogo. Em [Har89] é comentado mais exaustivamente este problema. [HS89] é um trabalho que se volta para a dificuldade de identificar o que é responsabilidade

da interface e o que é da aplicação. Para este trabalho é suficiente salientar a existência deste problema e que ele é ainda mais agravado quando se utiliza um modelo seqüencial de computação para o sistema interativo, pois não reflete a definição deste estilo em §3.2.

A arquitetura descrita na figura 5.4 para os componentes de uma sistema interativo, composto por processos que operam concorrentemente, parece mais realístico e trata-se de uma candidata a futuras implementações dos Estadogramas.

### 5.4.2 Diálogo *multithread*

Estadograma pode especificar diálogo *multithread* com simplicidade. Diálogo *multithread* oferece ao usuário várias tarefas que ele pode realizar e conduzir, sempre chaveando entre uma e outra. Representar esta situação em Estadogramas é feito de forma trivial usando-se estados AND, onde cada subestado é independente dos demais e opera concorrentemente. Quando as tarefas disponíveis em diálogos *multithread* podem ser realizadas paralelamente tem-se diálogos concorrente. Esta concorrência é tanto do ponto de vista do usuário quanto do sistema[Har89].

No capítulo 2 foi defendida a idéia de determinismo aplicado aos Estadogramas. Foram vistos várias vantagens para esta abordagem. Entretanto, isto inibe a especificação de diálogos concorrentes, como definido anteriormente. Isto porque, conforme [BG92], a coexistência de determinismo e concorrência em um mesmo sistema é praticamente impossível.

Diante desta situação, uma abordagem que parece promissora é permitir que o responsável pela especificação em Estadogramas defina como deseja ver interpretado os estados concorrentes. A opção por não determinismo permitiria a especificação de diálogos concorrentes.

## 5.5 Resumo

Viu-se, no capítulo anterior, informações pertinentes à construção do protótipo de uma interface desenvolvido neste trabalho. O controle do diálogo (sintaxe) da interface foi descrito com a notação dos Estadogramas. Este capítulo teceu observações sobre a notação dos Estadogramas para a especificação e implementação de diálogo. Parte delas decorrem da literatura referenciada e outra das dificuldades com o próprio desenvolvimento. Neste sentido este trabalho assemelha-se ao realizado por Wasserman[Was85].

As observações permitem concluir que ainda faltam recursos a serem acrescentados a esta notação para uso adequado para especificação de diálogo. Construções de mais alto-nível como *undo*; recursos para relacionar eventos lógicos e físicos; mudanças para melhorar a legibilidade de uma descrição na forma de texto, são alguns exemplos.

Do ponto de vista de implementação viu-se que um núcleo reativo para interfaces deve operar de forma concorrente e assíncrona com os demais componentes de um sistema interativo. Idealmente o *run-time* de um núcleo reativo deveria receber apenas dados para permitir mudanças imediatamente seguidas de testes, sem a necessidade de compilar código.

Parte das observações converteram-se em mudanças descritas no capítulo 2. Entretanto, este trabalho permite concluir que os desafios são grandes e só um trabalho maior pode enfrentá-los de forma satisfatória, principalmente pelo envolvimento de dois grandes temas: interfaces homem-computador e linguagens de especificação.

O próximo capítulo finaliza este trabalho, refaz concisamente os objetivos, motivação, o desenvolvimento e descreve as conclusões obtidas.

## Capítulo 6

# Conclusão

*“É bem conhecido que 99% dos problemas do mundo não são suscetíveis de solução por pesquisa científica. Há uma crença geral de que 99% da pesquisa científica não é relevante para os problemas do mundo real. Ainda assim todas as realizações e promessas da sociedade tecnológica moderna resultam da minúscula fração daquelas descobertas científicas que são, ao mesmo tempo, úteis e verdadeiras.”*

C.A.R. Hoare in foreword to  
*Systematic Software Development Using VDM.*  
C.B.Jonen. Prentice-Hall, 1986.

Viu-se o papel de destaque de uma interface na determinação do sucesso de um sistema interativo e seu caráter multidisciplinar. Desenvolver (projeto e implementação) uma interface consome parte significativa do tempo de desenvolvimento destes sistemas. Devido à abrangência do tema, este trabalho se restringe aos aspectos computacionais da interação homem-computador. Mais estritamente, na especificação e síntese do controle de diálogo.

Durante o projeto é necessário fazer o registro de várias informações, entre elas, o controle de diálogo da interface. Este controle é responsável por coordenar as atividades entre os componentes léxico (apresentação) e a aplicação (semântica). A figura 2.3 na página 20 mostra o relacionamento dos vários componentes de sistemas interativos.

Várias notações e modelos foram comentados para registrar o controle de diálogo (pág. 60). Contudo, não há um consenso acerca dos mais adequados. Ainda existem casos onde um ou outro se aplica de forma mais apropriada.

De maneira geral, os diagramas de transição de estados, ou melhor, extensões destes diagramas têm sido muito utilizadas para especificação do controle de interfaces. Estadogramas são extensões destes diagramas. Fornecem vários recursos adicionais tornando-os superiores aos diagramas de estados (§2.7.1). Da mesma forma que não se usam os diagramas de estados puros para especificação de interfaces[Was85], Estadogramas também precisam de “acréscimos” para torná-los adequados a esta atividade. Eles surgiram para descrição de

controle de sistemas reativos, aplicam-se a toda uma classe de sistemas com comportamento complexo e não exclusivamente a interfaces. Para o uso no âmbito de interfaces, algumas extensões devem ser previstas. O capítulo 5 forneceu várias alterações consideradas oportunas para o presente emprego.

Buscou-se identificar, durante o desenvolvimento de uma interface real (§4.3), as limitações que esta notação apresenta e, ao mesmo tempo, propor soluções para elas. Ainda procurou-se, baseado no atual estado do desenvolvimento de interfaces, selecionar as restrições a serem satisfeitas na implementação de uma notação candidata à especificação de diálogo. Os resultados são alterações descritas nos capítulos 2 e 5. Em §6.1 concentram-se as principais propostas sugeridas e as limitações deste trabalho.

O projeto de interfaces e questões correlatas ao desenvolvimento de interfaces foram comentados no capítulo 3. Relacionam-se com os objetivos deste trabalho. Um dos frutos do projeto é o comportamento (controle) da interface. Não se está interessado no quão adequado é o comportamento de uma interface para o usuário ou como pode ser obtido, mas na notação empregada para registrá-lo. Em outras palavras, fatores humanos estão além do presente interesse.

A implementação também foi considerada, pois a notação dos Estadogramas foi vista como linguagem executável. Esta consideração é imprescindível, pois a construção de protótipos é imperativa no desenvolvimento de interfaces (§3.5.6).

Tornar uma linguagem executável ou permitir que código possa ser gerado automaticamente implica em definir formalmente a semântica desta linguagem. No início do capítulo anterior foram feitos alguns comentários sobre a semântica formal dos Estadogramas. Em §2.6 são fornecidas várias referências que tratam da semântica formal desta notação.

Infelizmente não se teve acesso a outras ferramentas (exceto o Tradutor Blob) para o desenvolvimento da interface. Acredita-se que o uso de outras ferramentas possam solidificar ainda mais trabalhos com objetivos similares a este. Observar outras ferramentas, seus inconvenientes e qualidades, pode ser a melhor forma de obter informações valiosas e validadas pelo uso.

No capítulo 1 foram vistos vários motivos para “crer” nos benefícios dos Estadogramas para a especificação de diálogo. Entretanto, só a comparação com outras linguagens e ferramentas de apoio pode revelar as verdadeiras vantagens e desvantagens com relação aos métodos atualmente empregados. Contudo, isto só será possível após a construção de um “ambiente de qualidade.” Este termo refere-se a um produto (sistema) que forneça aos Estadogramas todo o apoio como têm outras notações empregadas. Isto inclui simuladores, geradores de código, verificadores para a sintaxe (por exemplo, evitar especificações onde estados jamais serão atingidos) e assim por diante. Enfim, investir nesta notação assim como outras já estabelecidas.

Investimentos deste porte envolvem grandes esforços. São necessários trabalhos “satélites” para fornecer informações que irão orientar um futuro projeto. Neste sentido este trabalho



pode ser visto como um “protótipo” de tal empreendimento, onde ainda estão sendo estabelecidos os requisitos.

Convém salientar que este trabalho procurou identificar características que pudessem melhorar o emprego desta notação neste contexto. Não se tem uma posição absoluta acerca dos benefícios das sugestões (capítulo 5) realizadas. Só o emprego mais exaustivo poderá torná-las recursos valiosos. Até o momento tem-se apenas indícios favoráveis.

O trabalho aqui realizado segue caminho semelhante às extensões que Wasserman[Was85] propõe para os diagramas convencionais. Em [Was85], Wasserman identifica as deficiências com estes diagramas, propõe extensões e só então as avalia. Seria ingênuo imaginar que as propostas naquele trabalho não sofreram todo um processo de mudanças. Só após o fim deste processo, que provavelmente refez várias vezes as extensões sugeridas, foram feitas as avaliações com base no ambiente que suportava as alterações. Neste trabalho há um impulso para um processo semelhante aplicado aos Estadogramas. Precisa-se ainda polir as sugestões e fornecer ferramentas que tornem atraente esta notação.

Espera-se que este trabalho tenha dado o primeiro passo neste sentido. As extensões propostas ainda precisam ser “solidificadas” com outros exemplos e circunstâncias a que se aplicam. No entanto, isto não invalida as sugestões. Lembremo-nos que Harel[Har87] sugeriu que só outros trabalhos forneceriam “ajustes” na notação que estava propondo. Seria mais coerente, portanto, imaginar que este trabalho forneceu *insights*, através das alterações propostas, e que precisam ser refinadas e incorporadas no que se chamou de “ambiente de qualidade.” Só então dar-se-á, com justiça, comparações com outras técnicas já exploradas há algum tempo, bem como afirmações mais seguras.

## 6.1 Contribuições da tese

O trabalho possui limitações. Alguns dos itens abaixo são sugestões para trabalhos futuros, não necessariamente são limitações pois encontram-se além dos objetivos deste trabalho. Um trabalho de maior envergadura, contudo, deverá considerá-las:

- Ausência de um tratamento formal para as alterações. Evitou-se discutir casos particulares e conflitos induzidos pelas alterações, contudo, deverão necessariamente ser considerados posteriormente. Esta situação é consequência, sobretudo, da dificuldade de tratamento formal desta linguagem[HPSS87].
- Mostrou-se para cada alteração apenas um exemplo que a justifica. Entretanto, mais exemplos aplicados a outras situações parece mais razoável e convincente.
- Apesar de vários trabalhos relacionarem o uso de diagramas de transição de estados como linguagem para uso por não programadores[Jac86], Estadogramas parecem possuir um comportamento mais complexo do que se espera para pessoas não versadas em computação. Entretanto, a tecnologia para o desenvolvimento de sistemas que podem

ser usados por não programadores está na sua infância e é uma área ativa de pesquisa da qual poucos produtos comerciais surgiram[DL91].

- Neste trabalho não foi considerado uma questão de muita importância: o desempenho da implementação dos Estadogramas. Houve uma ênfase maior na notação. A implementação, contudo, não pode desconsiderar este aspecto.
- Viu-se anteriormente o termo “ambiente de qualidade.” Este termo referia-se a um conjunto de ferramentas para dar suporte ao uso da notação dos Estadogramas. Este ambiente precisa ser construído. Note que Statemate[HLN<sup>+</sup>90] é visto aqui como um “ambiente de qualidade,” contudo, de uso e propósito geral. Está-se interessado no uso aplicado a especificação de diálogo. Nestas circunstâncias, naturalmente podem ser acrescentadas características peculiares ao âmbito de emprego, explorando a especificidade. Este ambiente deveria prover ferramentas, por exemplo, para detecção de estados que jamais serão atingidos, evitar *deadlocks* e assim por diante.

Apesar das restrições acima conseguiu-se vislumbrar nesta dissertação questões correlatas e o abrangente envolvimento deste emprego dos Estadogramas.

Foram identificadas várias necessidades de alterações na notação empregada bem como restrições a serem satisfeitas na implementação. Como visto anteriormente, esta é apenas a atividade inicial de um longo trabalho. No primeiro passo, contudo, espera-se ter atingido os objetivos deste trabalho.

Abaixo segue, resumidamente, as principais alterações consideradas propícias. O capítulo 5 comenta cada um dos itens abaixo:

- Ampliar a notação para suportar construções léxicas ou, alternativamente, fornecer este suporte de forma isolada do controle de diálogo como em [Sin89]. Integrar ferramentas para descrever estes dois aspectos, contudo, ainda são idéias em experimentação[BC91, pág. 29].
- Viu-se que o conceito de modo é muito sutil quando aplicado a interfaces que fazem uso de manipulação direta (§5.2.1). Há controvérsias neste sentido. O conceito de modo, ou melhor estado, precisa ser bem definido para orientar as especificações. Em §5.2.1 foram exemplificados dois usos de modo. Em um era modelado um processo, em outro um “estado” conforme percebido pelo usuário. Pensa-se que esta última opção seja a mais apropriada.
- Alteração dinâmica do comportamento, ou seja, “criação” e “remoção” de estados; alterar a origem e destino de transições, removê-las e outros conceitos parecem ser os equivalentes em Estadogramas dos tratadores de evento do modelo de eventos[Gre86]. Explorar estes conceitos pode aumentar o poder de expressão dos Estadogramas.
- *Undo*.

- Escopo de eventos. Permitir que contextos possam alterar a interpretação de um evento.
- Estabelecer prioridades para o tratamento de estados concorrentes em implementações determinísticas.
- Definir transição a ser seguida quando nenhuma outra se aplicar ao evento lógico gerado.
- Permitir que a implementação suporte a arquitetura de um sistema interativo como um conjunto de processos que operam de forma concorrente.

## 6.2 Trabalho futuro

Foram comentados vários itens a serem considerados em trabalhos futuros. Em vez de concentrá-los nesta seção, preferiu-se fornecer uma visão mais ampla que direta ou indiretamente abrangem todos eles.

Na literatura encontrou-se apenas um UIMS que fizesse o emprego de Estadogramas para especificação de diálogo[Wel89]. Esta referência, contudo, buscou uma implementação mais próxima da especificação fornecida em [Har87], diferente da abordagem deste trabalho. Pensa-se que uma evolução natural deste trabalho é considerar a construção de várias ferramentas para o suporte de Estadogramas aplicados a especificação de diálogo. Este conjunto é denominado na literatura de UIMS.

Estas ferramentas poderiam refletir as alterações aqui propostas e aquelas frutos da análise de outros sistemas com propósitos semelhantes.

## 6.3 Considerações finais

Marcus e van Dam[MvD91] afirmam que, num futuro próximo, haverá uma busca de cores, animação e som (possivelmente em três dimensões) em interfaces. Comentam que facilidade de aprender e usar irá requerer uma análise extensiva dos processos cognitivos do usuário, suas necessidades, desejos, habilidades e atitudes. Para isso, projetistas de comunicação visual, sociólogos, psicólogos e antropólogos terão que se unir a programadores, especialistas em fatores humanos e engenheiros.

Decorre do parágrafo anterior que interfaces homem-computador é uma área abrangente. Apenas uma fração diminuta dela é considerada neste trabalho. Mesmo para esta pequena fração os problemas não foram resolvidos.

O desenvolvimento realizado permitiu identificar a dificuldade de se empregar alguns conceitos, principalmente independência de diálogo. Ainda forneceu subsídios que deram

origem as propostas de mudanças nos Estadogramas, embora não da forma ideal. Faltaram outros sistemas para que pudesse ser verificado o que podem oferecer.

As propostas apresentadas seguramente podem ser melhoradas, seja do ponto de vista sintático ou semântico. Apesar de tudo, significam um progresso. Pensa-se que isto atinge os objetivos deste trabalho.

# Bibliografia

- [Abo91] Gregory D. Abowd. *Formal Aspects of Human-Computer Interaction*. PhD thesis, University of Oxford, University of York - Department of Computer Science — Heslington, York, Y01 5DD, England, June 1991.
- [AS91] David M. Andersen and Bruce A. Sherwood. Portability and the GUI. *Byte*, 16(12):221–226, November 1991.
- [Bac] Rodolfo Miguel Baccareli. *Micromundo Musical: Um Ambiente de Experimentação Auxiliado por Computador*. Tese de Mestrado. A ser publicada.
- [BB87] Ronald M. Baecker and William A. S. Buxton, editors. *Readings in Human-Computer Interaction - A Multidisciplinary Approach*. Morgan Kaufmann Publishers, Inc., 1987. In "Introduction".
- [BC91] Len Bass and Joëlle Coutaz. *Developing Software for the User Interface*. SEI Series in Software Engineering. Addison-Wesley Publishing Company, Inc., 1991.
- [BG92] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.
- [BMS86] Daniel G. Bobrow, Sanjay Mittal, and Mark J. Stefik. Expert Systems: Perils and Promise. *Communications of the ACM*, 29(9):880–894, September 1986.
- [Boe88] Barry W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, pages 61–72, 1988.
- [CH88] M. H. Chignell and P. A. Hancock. Intelligent interface design. In M. Hellander, editor, *Handbook of Human-Computer Interaction*, chapter 46, pages 969–995. Elsevier Science Publishers B.V., 1988.
- [Chi85] Uli H. Chi. Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches. *IEEE Transactions on Software Engineering*, SE-11(8):671–685, August 1985.

- [CMK88] John M. Carrol, Robert L. Mack, and Wendy A. Kellogg. Interface Metaphors and User Interface Design. In Martin Helander, editor, *Handbook of Human-Computer Interaction*. Elsevier Science - Publishers B.V, 1988.
- [Cou85] Joëlle Coutaz. Abstractions for User Interface Design. *IEEE Computer*, 18(09):21-34, September 1985.
- [Cur91] Bill Curtis. Japan's Research Focus Shifts to Interfaces. *IEEE Software*, November 1991.
- [DH89] Doran Drusinsky and David Harel. Using Statecharts for Hardware Description and Synthesis. *IEEE Transactions on Computer Aided Design*, 8(7):798-807, July 1989.
- [DL91] J. F. DeSoi and W. M. Lively. Survey and Analysis of Nonprogramming Approaches to Design and Development of Graphical User Interfaces. *Information and Software Technology*, 33(6):413-424, July 1991.
- [DN85] Stephen W. Draper and Donald A. Norman. Software Engineering for User Interfaces. *IEEE Transactions on Software Engineering*, SE-11(3):252-258, March 1985.
- [Eli92] Valéria Gonçalves Soares Elias. Editor Gráfico de Estadogramas, Dezembro 1992. DCC - IMECC - UNICAMP. Tese de Mestrado.
- [FB87] Mark A. Flecchia and R. Daniel Bergeron. Specifying Complex Dialogs in ALGAE. In *Proceedings of the ACM CHI+GI'87 Conference*, pages 229-234, Toronto, Canada, April 1987.
- [Fil91a] Antonio Gonçalves Figueiredo Filho. Geração de Gerenciadores de Sistemas Reativos. *V Simpósio Brasileiro de Engenharia de Software*, 1991. p'ags. 31-44.
- [Fil91b] Antonio Gonçalves Figueiredo Filho. Um Processo de Síntese de Sistemas Reativos, Dezembro 1991. DCC - IMECC - UNICAMP. Tese de Mestrado.
- [FvD82] J. D. Foley and A. van Dam. *Fundamentals of Interactive Computer Graphics*, chapter The Design of User-Computer Graphic Conversations, pages 217-243. Addison-Wesley Publishing Company, Inc., 1982.
- [FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, Inc., second edition, 1990. Chapters 8-10, about interfaces.
- [Gai86] Jason Gait. Pretty Pane Tiling of Pretty Windows. *IEEE Software*, pages 9-14, September 1986.

- [GG86] N. Gehani and A. D. Gettrick, editors. *Software Specification Techniques*. International Computer Science. Addison-Wesley Publishing Company, 1986.
- [GJM91] Carlos Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Printice-Hall, Inc., 1991.
- [Gre85] Mark Green. Report on Dialogue Specification Tools. In Günther E. Pfaff, editor, *User Interface Management Systems*, pages 9–20. Springer-Verlag, 1985.
- [Gre86] Mark Green. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5(3):244–275, July 1986.
- [GWWJ84] Michael D. Good, John A. Whiteside, Dennis R. Wixon, and Sandra J. Jones. Building a User-Derived Interface. *Communications of the ACM*, 27(10):1032–1043, October 1984.
- [Hab91] Frits Habermann. Giving Real Meaning to 'easy-to-use' Interfaces. *IEEE Software*, pages 90–91, July 1991.
- [Hal90] Anthony Hall. Seven Myths of Formal Methods. *IEEE Software*, pages 11–19, September 1990.
- [Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [Har88] D. Harel. On Visual Formalism. *Communications of the ACM*, 31(5):514–530, May 1988.
- [Har89] Rex Hartson. User-Interface Management Control and Communication. *IEEE Software*, pages 62–70, January 1989.
- [HGdR88] C. Huizing, R. Gerth, and W. P. de Roever. Modelling Statecharts Behaviour in a Fully Abstract Way. In *Lecture Notes in Computer Science*, pages 271–294. Springer-Verlag, 1988. Number 299.
- [HH89a] H. Rex Hartson and Deborah Hix. Human-Computer Interface Development: Concepts and Systems for its Management. *ACM Computing Surveys*, 21(1):5–92, March 1989.
- [HH89b] H. Rex Hartson and Deborah Hix. Toward Empirically Derived Methodologies and Tools for Human-Computer Interface Development. *Int. J. Man-Machine Studies*, pages 477–494, 1989.
- [Hil87] Ralph D. Hill. Event-Response Systems — A Technique for Specifying Multi-Thread Dialogues. In *Proceedings of the ACM CHI+GI'87 Conference*, pages 241–248, Toronto, Canada, April 1987. Special Issue of SIGCHI Bulletin.

- [Hix90] Deborah Hix. Generations of User-Interface Management Systems. *IEEE Software*, pages 77–89, September 1990.
- [HLN<sup>+</sup>90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, April 1990.
- [HLS90] Andrew Harbert, William Lively, and Sallie Sheppard. A Graphical Specification System for User-Interface Design. *IEEE Software*, 7(4):12–20, July 1990.
- [Hop91] Don Hopkins. The Design and Implementation of Pie Menus. *Dr. Dobb's Journal*, pages 16–26, December 1991.
- [HP85] D. Harel and A. Pnueli. On the Development of Reactive Systems. Technical Report, The Weizmann Institute of Science — Department of Applied Mathematics, Rehovot 76100, Israel, 1985.
- [HPSS87] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. *Proceedings of 2nd. IEEE Symposium on Logic in Computer Science*, pages 54–64, 1987.
- [HRdR92] J. J. M. Hooman, S. Ramesh, and W.P. de Roever. A Compositional Axiomatization of Statecharts. *To appear in TCS*, 1992. This research was supported by Esprit Project. Ramesh e-mail: ramesh@dhruv.ernet.in.
- [HS89] William D. Hurley and John L. Sibert. Modeling User Interface-Application Interactions. *IEEE Software*, pages 71–77, January 1989.
- [IK92] Daniel Ichbiah and Susan Knepper. *MICROSOFT*. Editora Campus, Rio de Janeiro, 1992. Tradução do original: *The Making of Microsoft*.
- [iLI87] i Logix Inc. The Languages of STATEMATE, 1987. Burlington, MA.
- [iLI89a] i Logix Inc. STATEMATE: Semantics of statecharts, August 1989. Burlington, MA.
- [iLI89b] i Logix Inc. The Statemate Approach to Complex Systems, August 1989. Burlington, MA.
- [Jac83] Robert J. K. Jacob. Using Formal Specifications in the Design of a Human-Computer Interface. *Communications of the ACM*, 26(4):259–264, April 1983.
- [Jac86] Robert J. K. Jacob. A Specification Language for Direct-Manipulation User Interfaces. *ACM Transactions on Graphics*, 5(4):283–317, October 1986.



- [KA90] Setrag Khoshafian and Razmik Abnous. *Object Orientation: Concepts, Languages, Databases, User Interfaces*, chapter User Interfaces, pages 323–387. John Wiley & Sons, Inc., 1990.
- [KK91] Tomihisa Kamada and Satoru Kawai. A General Framework for Visualizing Abstract Objects and Relations. *ACM Transactions on Graphics*, 10(1):1–39, 1991.
- [KP92] Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. *Lecture Notes in Computer Science*, 571:591–620, 1992.
- [KS89] Elieser Kantorowitz and Oded Sudarsky. The Adaptable User Interface. *Communications of the ACM*, 32(11):1352–1358, November 1989.
- [Lan88] Thomas K. Landauer. Research Methods in Human-computer interaction. In M. Helander, editor, *Handbook of Human-Computer Interaction*, chapter 42, pages 905–927. Elsevier Science - Publishers B.V, North-Holland, 1988.
- [LCV] Mark A. Linton, Paul R. Calder, and John M. Vlissides. InterViews: A C++ Graphical Interface Toolkit. Center for Integrated Systems/Stanford, CA 94305.
- [Mey85] Bertrand Meyer. On Formalism in Specifications. *IEEE Software*, pages 6–26, January 1985.
- [mGGW89] Limei Gilham, Allen Goldberg, and T. C. Wang. Toward Reliable Reactive Systems. *ACM SIGSOFT Engineering Notes*, 14(3):68–74, May 1989.
- [MLJ89] Bonnie E. Melhart, Nancy G. Levison, and Matthew S. Jaffe. Analysis Capabilities for Requirements Specified in Statecharts. *ACM SIGSOFT Engineering Notes*, 14(3):100–102, May 1989.
- [MN90] Rolf Molich and Jakob Nielsen. Improving a Human-Computer Dialogue. *Communications of the ACM*, 33(3):338–348, March 1990.
- [Moe90] Sven Moen. Drawing Dynamic Trees. *IEEE Software*, pages 21–28, July 1990.
- [MR92] Brad A. Myers and Mary Beth Rosson. Survey on User Interface Programming. In *Human Factors in Computing Systems CHI'92 Conference Proceedings*, pages 195–202, Monterey, California, May 1992.
- [MvD91] Aeron Marcus and Andries van Dam. User-Interface Developments for the Nineties. *IEEE Computer*, pages 49–57, September 1991.
- [Mye88] Brad A. Myers. A Taxonomy of Window Manager User Interfaces. *IEEE Computer Graphics & Applications*, pages 65–84, September 1988.

- [Mye89] Brad A. Myers. Encapsulating Interactive Behaviors. In *Human Factors in Computing Systems, Proceedings SIGCHI'89*, pages 319–324, Austin, TX, April 1989.
- [Mye92] Brad A. Myers. Demonstrational Interfaces: A Step Beyond Direct Manipulation. *IEEE Computer*, pages 61–73, August 1992.
- [Neu89] Peter G. Neumann. Flaws in Specification and What to do about Them. *ACM SIGSOFT Engineering Notes*, 14(3), May 1989.
- [Nie86] Jacob Nielsen. A Virtual Protocol Model for Computer-Human Interaction. *Int. J. Man-Machine Studies*, 24:301–312, 1986.
- [Nie92] Jacob Nielsen. The Usability Engineering Life Cycle. *IEEE Computer*, pages 12–22, February 1992.
- [Nor91] Kent L. Norman. Models of the Mind and Machine: Information Flow and Control between Humans and Computers. In Marshall C. Yovits, editor, *Advances in Computers*, pages 201–254. Academic Press, Inc., 1991.
- [NS79] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics*, chapter 28, pages 443–478. McGraw-Hill, Inc., second edition, 1979.
- [OGL<sup>+</sup>87] Dan R. Olsen, Mark Green, Keith A. Lantz, Andrew Schulert, and John L. Silbert. Whither (or wither) UIMS? In *Proceedings of the ACM CHI+GI'87 Conference*, pages 311–314, Toronto, Canada, April 1987.
- [Pin90] A. H. Pinon. Editor Topológico para a Linguagem de Especificação de Computações - LegoShell, 1990. DCC - IMECC - UNICAMP. Tese de Mestrado.
- [PS91] A. Pnueli and M. Shalev. What is in a Step: On the Semantics of Statecharts. In *Lecture Notes in Computer Science*, pages 244–264. Springer-Verlag, 1991. Number 526.
- [RXW92] Matthew D. Russell, Howard Xu, and Lingtao Wang. Action Assignable Graphics - A Flexible Human-Computer Interface Design Process. In *Human Factors in Computing Systems CHI'92 Conference Proceedings*, pages 71–72, Monterey, California, May 1992.
- [SB89] R. Summersgill and D. P. Browne. Human Factors: Its Place in System Development Methods. *ACM SIGSOFT Engineering Notes*, 14(3):227–234, May 1989.
- [SG86] Robert W. Scheifler and Jim Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

- [SG91] Gurminder Singh and Mark Green. Automating the Lexical and Syntactic Design of Graphical User Interfaces: The UofA UIMS. *ACM Transactions on Graphics*, 10(3):213-254, July 1991.
- [Shn83] Ben Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57-69, August 1983.
- [Shn87] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publishing Company, Inc., 1987.
- [Sin89] Gurminder Singh. *Automating the Lexical and Syntactic Design of Graphical User Interfaces*. PhD thesis, University of Alberta, Edmonton, Alberta, 1989.
- [SV91] H. W. Six and J. Voss. User Interface Development: Problems and Experiences. *Lecture Notes in Computer Science*, 555:306-319, June 1991.
- [TB85] P. P. Tanner and W. A. S. Buxton. Some Issues in Future User Interface Management System (UIMS). In Günther E. Pfaff, editor, *User Interface Management Systems*, pages 67-79. Springer-Verlag, 1985.
- [Was85] Anthony I. Wasserman. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Transactions on Software Engineering*, SE-11(8):699-713, August 1985.
- [WC91] Michael Wilson and Anthony Conway. Enhanced Interaction Styles for User Interfaces. *IEEE Computer Graphics & Applications*, 11(2):79-90, March 1991.
- [Wel89] Pierre D. Wellner. Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation. In *Human Factors in Computing Systems, Proceedings SIGCHI'89*, pages 177-182, Austin, TX, April 1989.
- [Woo90] William G. Wood. Application of Formal Methods to System and Software Specification. In *Proceedings of the ACM SIGSOFT International Workshop on Formal Methods in Software Development*, pages 144-146, May 1990.
- [WPSK86] Anthony I. Wasserman, Peter A. Pircher, David T. Shewmake, and Martin L. Kersten. Developing Interactive Information Systems with the User Software Engineering Methodology. *IEEE Transactions on Software Engineering*, SE-12(2):326-345, February 1986.
- [YS92] Tom Yager and Ben Smith. Is Unix Dead? *Byte*, 17(9):134-146, September 1992.

# Apêndice A

## Windows

Refere-se neste texto ao produto *Windows* da Microsoft. Neste texto é visto como um “Sistema de Janela” conforme taxonomia empregada em [BC91], embora a Microsoft o considere um sistema operacional. Uma visão geral do *Windows* pode ser encontrada em [KA90, págs. 334-350].

O *Windows* é uma extensão do ambiente MS-DOS. Destaca-se pelo ambiente gráfico que fornece aos seus usuários. A interface desenvolvida neste trabalho é sobre este ambiente.

Neste apêndice encontram-se algumas características deste sistema; uma visão geral do seu funcionamento; a relação com o paradigma de objetos e questões referentes à implementação de programas para execução neste ambiente. Por último são fornecidas referências para o programador deste ambiente.

---

### A.1 Caracterizando o ambiente *Windows*

- *Interface gráfica* com o usuário fornecendo janelas, menus, caixas de diálogo (*dialog boxes*) e controles (p. ex., botões) para uso pelas aplicações.
- *Fila de mensagens*. Cada aplicação possui uma fila associada na qual são colocadas todas as entradas (ações do usuário, eventos gerados por outras aplicações ou o pelo próprio *Windows*). Nem todas as mensagens são colocadas na fila, algumas são enviadas automaticamente para a aplicação.
- *Independência de dispositivos gráficos*. O *Windows* permite, com as mesmas funções, desenhar gráficos em um impressora matricial ou em um *display* de alta resolução, fornece independência de dispositivo.

- *Multitarefa (multitask)*, embora *non-preemptive*.<sup>1</sup>
- *Comunicação entre processos*. Permite a troca de dados entre aplicações.

## A.2 Visão geral do funcionamento

Para usufruir dos recursos fornecidos pelo *Windows* o programador C (principal linguagem de programação para aplicações *Windows*) conta com várias bibliotecas. Elas permitem realizar operações de entrada, saída, gerenciamento de memória e outras atividades compartilhando os recursos com outras aplicações. Compartilhar recursos obriga o abandono de rotinas tradicionais como `getchar`, `printf` e outras. No ambiente DOS um programa obtém informações através do teclado, por exemplo, fazendo uma chamada como `getchar`. No ambiente *Windows* todas as entradas do teclado, mouse, e *timer*, são interceptadas pelo *Windows* que as transfere para a aplicação a qual se destinam. Em *Windows* (ambiente multiprogramado) os programas devem compartilhar estes recursos e cooperarem entre si. Programas devem possuir uma estrutura que não monopolize o uso da CPU.

Uma aplicação compartilha a tela com outras aplicações usando uma janela para interagir com o usuário. Uma janela (veja nota de rodapé na página 4) é o dispositivo de entrada e saída de uma aplicação.

Uma vez que a aplicação recebe todas as entradas através do *Windows*, toda aplicação possui um “loop” para obtê-las. Tal “loop” encarrega-se de despachar para a rotina apropriada toda e qualquer informação existente para uma aplicação. A figura A.1 exemplifica este processo. Quando o usuário pressiona a tecla “A,” por exemplo, o *Windows* coloca tal informação na fila do sistema. A mensagem então é copiada para a fila da aplicação pertinente. O “loop” de mensagem da aplicação obtém a informação de que esta tecla foi pressionada e encarrega-se de entregá-la à rotina responsável por tratá-la. Uma aplicação pode possuir mais de uma janela e cada janela possui uma rotina que trata suas mensagens. Como resposta da aplicação executa a função `TextOut` para exibir na tela o caractere recebido.

## A.3 Paradigma de objetos e *Windows*

O uso do paradigma de objetos neste ambiente é nítida para o programador. Por exemplo, para criar uma janela deve ser definida uma classe especificando as características desejáveis. Só então instâncias dessas classes, as janelas, podem ser criadas. Várias aplicações podem

---

<sup>1</sup>Em sistemas operacionais multitarefa é necessário uma política para selecionar o processo a ocupar a CPU. Quando o sistema é *non-preemptive* o processo em execução pode monopolizar o uso da CPU. Em um sistema *preemptive* o processo em execução pode ser interrompido e a política de escalonamento pode ser aplicada. No caso anterior é preciso contar com a colaboração dos processos em execução. Veja William Stallings, *Operating Systems*, Macmillan Publishing, 1992, pág. 359.

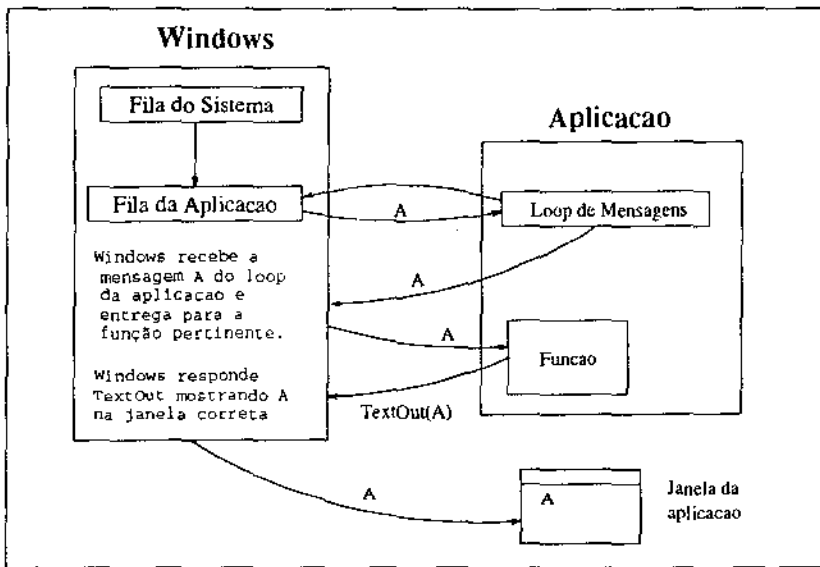


Figura A.1: Triagem de mensagens no *Windows*.

compartilhar a mesma definição de uma janela simultaneamente. A comunicação com as janelas é realizada via envio e recebimento de mensagens.<sup>2</sup>

## A.4 Construindo aplicações *Windows*

Construir programas para o ambiente *Windows* sem apoio de ferramentas significa realizar chamadas a rotinas com até 15 argumentos; dominar a semântica de centenas de mensagens e da relação entre elas e o ambiente; procurar entre mais de mil rotinas a que se deseja; gerenciar um razoável número de variáveis globais e gerenciamento de memória peculiar ao ambiente.

Programar uma aplicação *Windows* consiste em uma atividade recheada de oportunidades de erro. A solução é possuir um nível de abstração sobre os gerenciadores de janela (§3.4). Por outro lado, abstrações fornecidas pelo *InterViews*, por exemplo, exigem um programador com habilidade suficiente no paradigma de objetos e em C++.

A figura A.2 permite identificar os arquivos e utilitários necessários durante a confecção de uma aplicação *Windows*. Além do código pertinente à aplicação, i.e., arquivos *.c*, *.h* e *.asm* existem outros arquivos. Aqueles com a extensão *.ico*, *.cur* e *.bmp* definem respectivamente os ícones, os tipos de cursor e figuras utilizados. Tais arquivos podem ser gerados por utilitários fornecidos com o SDK (*Software Development Kit*). Arquivos *.dlg* descrevem caixas de diálogo. Arquivos *.fnt* contêm fontes (estilos de letras), que também podem ser

<sup>2</sup>Eventos gerados pela ação do usuário sobre dispositivos de entrada ou informações trocadas entre processos.

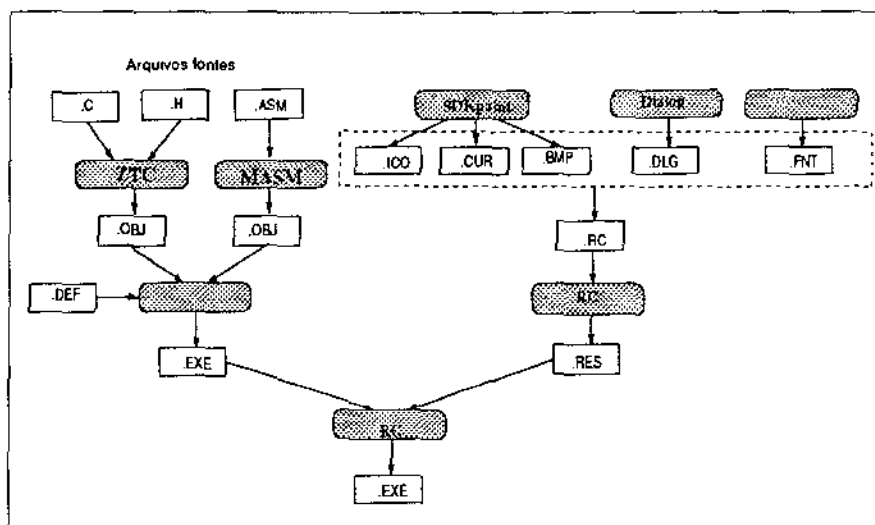


Figura A.2: Construindo uma aplicação *Windows*.

geradas usando o SDK.

Cursor, ícones, diálogos e outros recursos são reunidos no arquivo `.rc`. Um outro arquivo especial, `.def`, descreve alguns detalhes da aplicação. Este arquivo é necessário na fase de ligação dos módulos objetos – fornece informações sobre a aplicação (organização da memória, por exemplo). Após esta ligação, o resultado da compilação do arquivo `.rc` (arquivo `.res`) é ligado ao arquivo `.exe` através do utilitário `RC` (acompanha o SDK). Após este último passo tem-se um programa *Windows*.

## A.5 Referências para o programador

1. O manual *Guide to Programming do Microsoft Windows Software Development Kit (SDK)* fornece uma visão geral e os primeiros passos no desenvolvimento de aplicações *Windows*. As figuras deste apêndice foram retiradas deste manual.
2. A “bíblia” do programador *Windows*, contudo, é o livro de Charles Petzold, *Programming Windows 3.1*, Microsoft Press, 1992.
3. *Windows 3.1: A Developer's Guide, 2nd edition*, Jeffrey M. Richter, 1992, contém características mais sofisticadas. Não é um livro introdutório.
4. *Graphics Programming Under Windows*, SYBEX, Brian Meyers e Chris Doner, 1988, é uma excelente fonte para quem quer saber mais detalhes sobre síntese de imagens neste ambiente.

5. Segredos do Windows 3.1, Berkeley Brasil Editora, Brian Livingstone, 1992, fornece informações para gerência do ambiente *Windows 3.1*.
6. *The Windows Interface – An Application Design Guide*, Microsoft Press, 1992. Este livro é voltado para as últimas atividades do projeto de interfaces, onde o projetista identifica botões, menus, caixas de diálogo. O livro fornece uma visão geral destas atividades e auxilia a implementação delas.
7. *The Microsoft Guide to C++ Programming*, Microsoft Press, Kaare Christian, 1992. Fornece uma introdução à linguagem C++ ao mesmo tempo que exemplifica e instrui sobre o uso de um conjunto de classes. Estas classes formam uma camada orientada a objetos sobre a interface de programação do *Windows* fornecida junto com o compilador da *Microsoft*. São conhecidas por *Microsoft Foundation Class* (MFC).



# Índice

- ação, *veja* eventos, 24
- abrangência, 8
- Ada, vii, 33
- alteração dinâmica, 91
- alterações, 22
- ambiente, 79
  - de desenvolvimento, 79
- ambientes gráficos, 4
- análise, 83–101
- Apple Macintosh*, 4
- apresentação, 65
- aresta, 25
- arquitetura de software, 48, 49
- Arquivista, 69
- Assistente, 69
- atividade, 24
- avaliação
  - da interface, 55
- C
  - linguagem, 118
  - callback function*, 87
  - cognitivo, *veja* modelo
  - comunicação entre processos, 118
  - conceitos, *veja* interface
  - conceitual, *veja* modelo
  - conclusão, 103–108
  - consistência, 51
  - construtor de interface, *veja interface builder*
  - controlador de diálogo, 50
  - controle de diálogo, 10
  - convenções, 14
- descrição dos estadogramas, 21
- desenvolvimento, 67–82
  - ambiente, 79
  - da interface, 67
- device driver, 49
- diálogo, 41, 52
- Editor, 69
- Egrest, 34
- escopo, 8
- escopo de eventos, 92
- especificação
  - dificuldades com, 60
- Estadograma
  - análise de, 83
- Estadogramas, 14–39
  - descrição de, 20
  - O que são?, 20
  - referências, 21
- estados
  - concorrentes, 23
  - ortogonais, 23
- estilos, 44
  - form fill-in*, 45
  - icônico, 45
  - linguagem natural, 44
  - linguagem de comandos, 44
  - manipulação direta, 45
  - menus, 45
- evento, 23
  - ação, 24
- eventos
  - ação, 21
  - atividades, 21
- Executor, 69

- expansões, *veja* trabalhos futuros
- feedback*, 51
- ferramentas, 32
- Egrest, 34
  - Reacto, 33
  - Statemaster, 33
  - Statemate, 33
  - Tradutor Blob, 34
- fila de mensagens, 117
- form fill-in*, 45
- fronteiras, 8
- hardware, 80
- hipótese, 30
- history*, 26
- ícone, 4, 45
- definição de, 45
- independência
- de diálogo, 52
  - de dispositivo, 117
  - de plataforma, 55
- independência de diálogo, 98
- como obter?, 54
- integração, 65
- interação, *veja* modelo
- estilos, 44
- Interface, 41-66
- interface, 42
- abrangência, 5
  - amigável, 2
  - arquitetura de software, 48, 49
  - conceitos, *veja* terminologia básica
  - custo, 3
  - definição, 1, 42
  - descrição, 70
  - especificação de, 61
  - estilos, 44
  - fácil de usar e aprender, 2
  - flexível, 54
  - gerenciadores, 78
  - homem-computador, 41
  - importância comercial, 4
  - portabilidade de, 55
  - tamanho, 3
  - terminologia básica, 41
- interface builder*, 63
- interface gráfica, 117
- InterViews*, 65, 119
- Introdução, 1-14
- janelas, 4
- sobrepostas, 4
- justificativas, 12
- LEG, 34
- limitações, *veja* conclusão
- limites, *veja* abrangência
- linguagem natural, 44
- linguagem de comandos, 44
- look and feel*, 44
- Macintosh, vii
- manipulação direta, 45, 98, 99
- melhoramentos, *veja* trabalhos futuros
- menu, 14
- menus, 45
- metáfora, 52
- desktop*, 52
- MM, 67
- micromundo musical, 67
- modelo
- cognitivo, 46
  - conceitual, 46
  - de interação, 46
  - de Seeheim, 10, 48
- modo, 84
- motivação, 12
- mouse, 14
- multitarefa, 117
- núcleo funcional, 50
- Notador, 69
- objeto de interação, 50
- orientação a objetos, 65

- paradigma de objetos, 65
- portabilidade, *veja* interface
- princípios, 39
  - de projeto, 51
- princípios de projeto, 50
- prioridades, 94
- problemas, 5
- projeto, 65, 76
  - princípios, 50
  - princípios de, 51
- protótipos, 55
- qualidade, 51
- rótulo, 25
- Reacto, 33
- reativo, 16
- reativos, *veja* sistemas
- resumo dos capítulos, 13
- retroalimentação, 51
- reutilização, 90
- run-time*, 19, 34
- SDK, 119
- sec-ambiente, 79
- Seeheim, *veja* modelo, *veja* modelo
- sistema, 67
  - de janela, 49
  - descrição, 67
  - reativo, 16
  - transformacional, 16
- sistemas
  - de janela, 43
  - reativos, 16
    - divisão, 19
- Smalltalk*, 4
- Smalltalk*, vii
- software, 80
  - arquitetura, 48, 49
- Statemaster, 33
- Statemate, 33
- Supervisor, 69
- taxonomia
  - sistemas reativos, 19
- terminologia básica, 41
- termos
  - uso, 14
- tese, 7
- toolkit*, 62
- trabalho futuro, 107
- Tradutor Blob, 34
- transformacional, 16
- transição
  - aresta, 25
  - rótulo, 25
- transição alternativa, 97
- transições, 95
- UIMS, 63, 78
  - Statemaster, 33
- undo*, 91
- usuário, 1
- variáveis, 89
- visão geral, 13
- WIMP*, 9
- Windows*, 4
- Windows*, vii, 117–121
- Windows NT*, 4
- WYSIWYG, 44