UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Representação em Projeto de Interfaces Homem-Computador: Estudo, Aplicação e Propostas de Extensão do Formalismo UAN

Autor: **Elton José da Silva**

Orientadora: **Profª. Drª. Beatriz Mascia Daltrini**

Dissertação submetida à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica.

julho/1995

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Si38r

Silva, Elton José da
Representação em projeto de interfaces homem-computador: estudo, aplicação e propostas de extensão do formalismo UAN. / Elton José da Silva.--Campinas, SP: [s.n.], 1995.

Orientador: Beatriz Mascia Daltrini.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Interfaces de usuário (Sistema de computador). 2. Interação homem-máquina. 3. Sistemas de computação interativos. 4. Usuário final. 5.* Interação homem-computador. I. Daltrini, Beatriz Mascia. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. III. Título.

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Curso de Pós-Graduação em Engenharia Elétrica da Universidade Estadual de Campinas, pela Comissão formada pelos professores:

Orientadora: Profª. Drª. Beatriz Mascia Daltrini (FEE-Unicamp)

Prof. Dr. Mario Jino (FEE-Unicamp)

Prof. Dr. Ivan L. M. Ricarte (FEE-Unicamp) (Suplente)

Profª. Drª. Agma Juci Machado Traina (ICMSC-USP)

Campinas, 25 de julho de 1995

Observei o conjunto da criação de Deus e percebi que
o homem não consegue entender toda a obra que se faz debaixo do sol.
Por mais que se afadigue em pesquisar, não chega a compreendê-la,
e mesmo que o sábio diga que a conhece, nem por isso é capaz de entendê-la.
Eclesiastes 8:17

Dedicatória

Àquele que nos traz a Vida e a Esperança, Jesus Cristo.
Aos meus pais, Dalva e Juarez, com amor e admiração.

Agradecimentos

Os últimos três anos foram realmente incríveis... Eu agradeço especialmente a Deus por mais essa etapa vencida. O meu *Muito Obrigado* também a todos os que participaram, contribuíram e incentivaram a realização deste trabalho.

- À amiga e orientadora Bia, pela oportunidade, paciência, incentivo e atenção infindáveis.
- Aos membros da Banca, Prof. Mario Jino, Profa. Agma e Prof. Ivan, pela participação. De forma especial gostaria de agradecer ao Prof. Mario Jino pela revisão acurada do texto.
- Ao Professor, sobretudo amigo, José Luís Braga, da UFV (Viçosa-MG), pelos *conselhos eletrônicos* e por me iniciar na pesquisa científica.
- À Profa Ting (DCA), pela dedicação com que faz o seu trabalho e por despertar em mim o desejo de continuar na área acadêmica.
- A várias pessoas que se envolveram diretamente com este trabalho, entre elas: Fábio Lucena (DCC), cujo trabalho de mestrado me serviu de incentivo; Craig Struble (*Virginia Tech*), por fornecer a ferramenta *QUANTUM* (cap. 6); Luiz Gonzaga (DCA), pelas horas gastas me ajudando a instalá-la e por sanar as minhas dúvidas em relação à modelagem geométrica; Perla (DCA), por me ajudar na especificação do *ProSIm*; Nelson, pela revisão do *Abstract* e dicas na organização da dissertação; Ladislau, pela prontidão em sempre ajudar, principalmente com o LaTeX; Plínio Vilela, pela leitura e correções na versão preliminar; Simone Scarpelini (CTI), Nilton (CPqD), Cecília (DCA) e Oscar (PUCCAMP), pelas discussões proveitosas e dicas sempre tão valiosas.
- De forma muito especial gostaria de agradecer a toda minha família, que constantemente me apóia e encoraja a aceitar novos desafios.
- Aos irmãos Nelson, Perla, Márcio Leandro, Giba, Wagner, Carlos e Rosana, com quem eu tenho tido a oportunidade de aprender e experimentar do verdadeiro amor de Cristo. Vocês são realmente uma benção em minha vida!

- A todos os amigos que sempre estiveram por perto: ao pessoal do LCA, às meninas da Engenharia de Alimentos, à turma do vôlei, ao pessoal do grupo de oração. À Olga (DCA), por me lembrar que *concluir a tese é também uma questão de responsabilidade.*
- Aos colegas da Moradia Estudantil, pela convivência quase sempre tranquila.
- À Angelita, Rodrigo, Dona Carmem, Seu Otávio e Vovó Niza, minha família adotiva em Campinas. Obrigado pelo carinho e pelos lanches deliciosos. Eu não teria sobrevivido sem eles!
- Aos alunos, professores e funcionários do Departamento de Computação da UFOP (Ouro Preto-MG), por "segurarem as pontas" durante o período I/95, e por reconhecerem a necessidade desta realização.
- Aos amigos das Repúblicas *TX*, em Ouro Preto, e *CR*, em Viçosa, pela hospitalidade.
- À CAPES, pelo auxílio financeiro e pela lição de sobrevivência.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

À medida que as interfaces com o usuário tornam-se mais fáceis de aprender e usar, elas também se tornam mais difíceis de especificar e programar de forma clara. Projetistas de interfaces geralmente usam técnicas informais ou *ad hoc* na fase de especificação. Essas técnicas, que são incompletas ou ambíguas, fazem com que os desenvolvedores interpretem as especificações de forma diferente. Isto pode levar a várias inconsistências e mal-entendidos que se propagam para os estágios finais de integração do sistema. Com o objetivo de reduzir esses problemas, existe hoje uma grande preocupação em se produzir especificações mais precisas e consistentes para comunicar a aparência e funcionalidade da interface entre os membros da equipe de desenvolvimento. Neste trabalho é analisada uma variedade de técnicas para representar o projeto da interface com o usuário, sendo que a técnica *UAN* (*User Action Notation*) destacou-se a princípio como a mais apropriada. Portanto, concentramos os nossos esforços no seu estudo, aplicação e validação. Algumas das razões para esta escolha é que *UAN* é simples, concisa, centrada nas ações e tarefas do usuário, e apropriada para descrever interfaces de manipulação direta. A *UAN* foi utilizada para especificar algumas interações numa interface de modelagem geométrica, como um meio de verificar o poder de expressão da notação. Apresentam-se os resultados dessa aplicação, enfatizando características atrativas e limitações encontradas na *UAN*, e algumas extensões que podem melhorá-la são sugeridas.

Palavras-chave: interação homem-computador, sistemas centrados no usuário, projeto de interfaces, técnicas de especificação, descrição de tarefas, *UAN* (*User Action Notation*).

Abstract

As user interfaces become easier to learn and use, they also become more difficult to specify and program clearly. Interface designers often use informal or *ad hoc* techniques for defining interfaces. These techniques, which are incomplete or ambiguous, cause developers to interpret the specifications differently. It might lead to the generation of inconsistencies and misinterpretations which may propagate to later stages of system integration. In this way, there is a strong push to produce more precise and consistent specifications to communicate the look and functionality of the interface among the members of the development team. This work analyzes a variety of existent techniques to specify user interfaces. *UAN* (*User Action Notation*) appears to be the most appropriate technique for the established purposes; therefore, the efforts are concentrated on its study, application and validation. Some of the reasons for this choice is that *UAN* is simple, concise, centered on user actions and tasks, and appropriate to specify direct manipulation interfaces. As a way of verifying *UAN* 's power of expression, it is applied to the specification of interactions in a geometric modelling user interface. The results of this application, emphasizing attractive features and limitations of the notation, are presented, and some extensions to improve it are suggested.

Keywords: human-computer interaction, user-centered systems, interface design, specification techniques, task description, *UAN* (*User Action Notation*).

Convenções e Terminologia

Na redação do texto desta dissertação, adotaram-se algumas convenções, a saber:

- Os termos usados na língua de origem são destacados em *itálico*. Entretanto, alguns termos, por serem considerados bastante usuais da Ciência da Computação, não aparecerão destacados, por exemplo, software, hardware, feedback, mouse, layout e menu.

- *HCI* (*Human-Computer Interaction*) engloba muitos termos utilizados frequentemente pela maioria das pessoas que trabalham na área. Os iniciantes geralmente sofrem com tantos termos, geralmente em inglês (por exemplo, *widget*, *touchscreen*, *groupware*), dificultando a compreensão. No corpo desta dissertação, o destaque de um termo em *itálico sublinhado* indica que uma definição sucinta encontra-se no glossário apresentado no final desta dissertação.

- Considera-se *HMI* (*Human-Machine Interaction*) uma área mais abrangente que estuda a interação entre o homem[1] e qualquer tipo de máquina durante a execução de uma determinada tarefa. *HCI* (*Human-Computer Interaction*) é considerada como uma subárea de *HMI*, para o caso particular de computadores.

- Idealmente, os termos *diálogo homem-computador* e *interface homem-computador* (também conhecido como interface com o usuário) são definidos separadamente para denotar, respectivamente, a troca de ações e símbolos na comunicação homem-computador e o suporte de hardware e software através do qual esta troca ocorre. Entretanto, na maioria da literatura sobre desenvolvimento de interfaces, ambos os termos são usados como sinônimos [Hartson89].

- Na Ciência da Computação, o termo *interface* possui muitos significados, dependendo do contexto em que é aplicado. Nesta dissertação, o termo *interface do computador com o usuário* muitas vezes aparece simplesmente como *interface do computador*, *interface com o usuário*, *interface homem-computador*, ou simplesmente *interface*. Para maiores detalhes sobre terminologia em *HCI*ver [Grudin93].

[1] Quando não estiver explicitado, a palavra *homem* engloba tanto homens quanto mulheres.

Capítulo 1

Introdução

O importante não é a tecnologia em si,
mas a forma como ela é apresentada aos seus usuários.
Don Valentine

Neste capítulo, define-se o termo *interface homem-computador*. Discute-se a sua crescente importância no desenvolvimento de sistemas interativos e alguns problemas normalmente encontrados nessa tarefa. Apresentam-se as motivações para realização deste trabalho, bem como os objetivos e escopo do mesmo. Apresenta-se também uma visão geral dos capítulos que compõem esta dissertação.

1.1 Interface: conceito, importância e problemas

Quando o conceito de *interface homem-computador* começou a surgir (década de 50), associava-se interface ao hardware e software através dos quais homem e computador podiam se comunicar. Naquela época, uma boa interface poderia ser, por exemplo, aquela que levasse os usuários (os cientistas da época) a manipular o menor número de válvulas, alavancas e botões. Daquela época até hoje houve uma grande evolução. As interfaces gráficas com o usuário (*GUI's*) constituem um dos avanços mais revolucionários na área de *HCI*[1]. No espaço de menos de dez anos a interação entre o usuário e o computador mudou de um simples diálogo baseado numa troca de caracteres alfanuméricos para as conhecidas interfaces baseadas em janelas, ícones, menus e síntese de voz.

[1] *Human-Computer Interaction.*

Hoje, considera-se como sendo interface não só o meio físico que separa duas entidades distintas – homem e computador – em seu processo de comunicação; ela funciona como um mediador que suprime as deficiências de cada entidade ao longo da interação. Assim, a interface não deve possuir apenas coerência visual mas, principalmente, dispositivos que aliviem a carga cognitiva do usuário [Laurel90] envolvendo, portanto, conhecimento sobre a maneira como as pessoas resolvem problemas.

Poder-se-ia ainda definir *interface homem-computador* como sendo a parte de um sistema interativo responsável por traduzir ações do usuário em ativações das funcionalidades do sistema, permitindo que os resultados possam ser observados e a interação devidamente coordenada.

Com o aumento do número de usuários e aplicações, as interfaces de computador com o usuário têm se tornado cada vez mais importantes. A revolução causada pelo computador pessoal e a queda dos preços do hardware tornaram o computador disponível para um grande número de pessoas, das mais diversas áreas de aplicação. Inicialmente, quando o computador era usado somente por um grupo restrito de poucas pessoas, na execução de tarefas especializadas, não era estranho que requisitasse grande experiência do usuário. Também naquela época, devido ao computador ser tão caro, os usuários poderiam "sofrer" um pouco em favor da eficiência computacional. Nos dias atuais, uma grande parte dos recursos computacionais é dedicada exclusivamente para tornar mais fácil a interação com o usuário. A *comunicação* com o sistema se tornou pelo menos tão importante quanto a *computação* feita pelo sistema. Essa característica é fundamental para que os usuários possam interagir eficientemente com o sistema a fim de tirar proveito da capacidade computacional. Os usuários têm se tornado cada vez mais exigentes e descartam qualquer interface que seja difícil e desconfortável de usar pois, para eles, a interface final é o sistema.

É constatado que, na maioria dos sistemas interativos atuais, uma interface pode consumir até cerca de 70% dos custos totais de desenvolvimento. Entretanto, do ponto de vista de muitos engenheiros de software, a interface ainda não é considerada como parte integral do sistema interativo como um todo, mas somente como uma característica a ser pensada depois. A ênfase emergente em usabilidade tenta mudar essa perspectiva, tornando a interface uma parte crítica do sistema interativo e, o seu desenvolvimento, considerado como parte integral do processo de engenharia de software.

1.2 Motivação

O Grupo de Interfaces com o Usuário (GIU), em formação no Departamento de Computação e Automação (DCA) da FEE/Unicamp, coordenado pela Profa. Beatriz Mascia Daltrini, tem investigado a área de Interação Homem-Computador, principalmente sob o prisma da Engenharia de Software. Por ser um grupo recente, trabalhando numa área relativamente nova da Computação, foram várias as idéias iniciais de trabalhos a serem desenvolvidos.

As principais motivações que impulsionaram o desenvolvimento deste trabalho específico podem ser resumidas em:

- Interesse em melhorar o *processo* de desenvolvimento de sistemas interativos. Os métodos convencionais da Engenharia de Software não têm contemplado de forma satisfatória o desenvolvimento da interface com o usuário. Acredita-se que enquanto a maneira como é feita esse desenvolvimento não for melhorada e bem estabelecida, interfaces com baixa *usabilidade*[2] continuarão a aparecer no mercado.

- A fase de projeto de interfaces, dentro do ciclo de desenvolvimento é considerada a mais difícil e criativa, e as técnicas atualmente empregadas para representação do projeto apresentam problemas.

- Percepção de que a técnica de especificação *UAN* (*User Action Notation*) poderia ser uma boa notação para descrever projetos de interface.

- Cooperação com o Grupo de Computação e Imagens (GCI), do DCA, trabalhando na especificação de uma interface para um sistema de síntese de imagens.

1.3 Objetivos

Os objetivos deste trabalho de pesquisa são:

- Entender o *processo* de desenvolvimento de sistemas interativos, principalmente a fase de projeto, do ponto de vista da Engenharia de Software.

- Enfatizando a atividade de projeto de interfaces, selecionar técnicas existentes para sua representação, classificá-las e compará-las, apresentando principais vantagens e

[2] do termo em inglês, refere-se à facilidade de utilização de um sistema interativo.

limitações de cada uma. Escolher uma dessas técnicas (ou uma combinação delas) para ser utilizada em especificações de interfaces a serem desenvolvidas futuramente pelo grupo.

- Validar a técnica escolhida, através de sua aplicação na descrição de uma interface real, provendo uma descrição precisa da interface, na forma como ela deverá ser percebida pelo usuário final.

- Propor extensões com o objetivo de facilitar a utilização e aumentar o poder de expressão da notação utilizada.

- Especificar uma ferramenta automática baseada na na notação utilizada, que apóie a tarefa de descrição do projeto da interface.

1.4 Estabelecimento de fronteiras

Apresentam-se a seguir algumas fronteiras para o escopo deste trabalho:

- Quando se usa o termo *interface*, refere-se a interfaces de manipulação direta *WIMP*[3], ou, como são mais conhecidas, *GUIs* (*Graphical User Interfaces*). Especificação de interfaces que envolvem novos meios de interação como linguagem natural, movimento dos olhos, ou associadas a *groupware*, *realidade virtual*, não foram consideradas durante o desenvolvimento deste trabalho.

- Sabe-se que a área de desenvolvimento de sistemas interativos é uma área multidisciplinar. Neste trabalho não são feitas maiores considerações sobre psicologia, ergonomia e aspectos sociais envolvidos nessa área. O interesse está nos aspectos computacionais do desenvolvimento de sistemas interativos, principalmente a engenharia de software envolvida em tais tipos de desenvolvimento. Mais especificamente, o trabalho envolve a representação do projeto da interface de um sistema interativo.

1.5 Visão geral da dissertação

O restante desta dissertação está organizado em seis capítulos, e quatro apêndices, descritos resumidamente a seguir:

[3] *Windows, Icons, Menus and Pointers.*

No **Capítulo 2** são apresentadas algumas noções gerais do desenvolvimento de sistemas interativos. O principal enfoque neste capítulo é a engenharia de software envolvida no processo de desenvolvimento de tais sistemas. Algumas abordagens de desenvolvimento de software são discutidas, comparando alguns dos modelos tradicionais e explorando um ciclo de vida que contempla especificamente o desenvolvimento da interface com o usuário.

No **Capítulo 3** são apresentadas algumas técnicas de representação utilizadas para especificar o projeto da interface de sistemas interativos. Antes disso é discutida uma possível classificação para essas técnicas, o problema da especificação de interfaces de manipulação direta, diálogos seqüenciais e assíncronos, interfaces modais e não-modais. Apresentam-se algumas das técnicas que aparecem na literatura, enfatizando as principais características, alguns exemplos, vantagens e problemas na utilização de cada uma delas.

No **Capítulo 4** o principal objetivo é fornecer uma base para o entendimento de descrições *UAN* de interfaces. Esta foi a notação básica utilizada para especificação da interface a ser discutida neste trabalho. São fornecidos um breve histórico, origem, descrição da notação e alguns exemplos de sua aplicação.

No **Capítulo 5** discute-se a aplicação da notação *UAN* na descrição de parte da interface do *ProSIm* (sistema em desenvolvimento pelo Grupo de Computação e Imagens do DCA). São feitas críticas analíticas à notação e propostas algumas alterações para facilitar a sua utilização e aumentar o seu poder de expressão.

No **Capítulo 6** descreve-se sucintamente a ferramenta automática *QUANTUM*, utilizada durante a especificação da interface do *ProSIm*.

No **Capítulo 7** apresentam-se as principais conclusões, contribuições do trabalho realizado e sugestões para trabalhos futuros.

No **Apêndice A** apresenta-se parte da documentação gerada pelas especificações *UAN* realizadas para a interface de modelagem geométrica do sistema *ProSIm*.

No **Apêndice B** discute-se resumidamente a linguagem *Tcl/Tk*, utilizada na implementação da ferramenta *QUANTUM*. Também são apresentados trechos de alguns programas-fonte que implementam a ferramenta.

No **Apêndice C** são apresentados alguns exemplos dos arquivos (*.qtm*) gerados na descrição do projeto da interface do *ProSIm*, utilizando a ferramenta *QUANTUM*.

Finalmente, no **Apêndice D**, apresenta-se um **Glossário** com alguns termos comuns da área de interação homem-computador.

Capítulo 2

Interação Homem-Computador: Uma Visão Geral

Homem e Computador são considerados dois
processadores de informação altamente sofisticados,
mas quando interagem, ocorrem sérias limitações de comunicação.
B. Shackel

Neste capítulo são apresentadas algumas noções gerais do desenvolvimento de sistemas interativos. Para uma abordagem mais completa do assunto, recomendam-se [Hartson89, Bass91, Hix93] como leituras adicionais. Devido a esta ser uma área relativamente nova, muito abrangente e ainda não possuir fronteiras bem estabelecidas, a literatura ainda está bastante dispersa; a maioria dos livros geralmente enfoca um ou outro aspecto relacionado ao desenvolvimento de sistemas interativos.

O principal enfoque neste capítulo é a engenharia de software envolvida no processo de desenvolvimento de sistemas interativos. Inicialmente são discutidas as principais subáreas relacionadas ao desenvolvimento de tais tipos de sistemas, bem como as principais profissionais e funções envolvidas. Apresenta-se uma comparação entre as diversas fases do processo de desenvolvimento do software da aplicação, as fases envolvidas no desenvolvimento do software da interface com o usuário e como essas duas atividades podem ser levadas paralelamente. Finalmente, são apresentadas algumas abordagens de desenvolvimento de software, comparando alguns dos modelos tradicionais e explorando um ciclo de vida que contempla especificamente o desenvolvimento da interface com o usuário.

2.1 Um pouco de história

A preocupação com a chamada *engenharia de fatores humanos*[1] começou durante a Segunda Guerra Mundial. Na década de 40, os pilotos da Força Aérea Americana tiveram muitos acidentes atribuídos a *erros do piloto.* O governo americano contratou, então, um grupo de psicólogos para estudar os pilotos e descobrir o que havia de errado com eles. Depois de testes e observações, os psicólogos concluíram que não havia nada de errado com os pilotos, o problema era com o projeto dos aviões.

Por muitos anos, as pessoas foram condicionadas a atribuir a maioria dos acidentes a *erros humanos*, e levou-se muito tempo para descobrir que estes erros poderiam ser causados pela maneira como se projetava o equipamento que estava sendo utilizado. Naquela época, o piloto voava em vários tipos de aviões existentes. Havia então muitas inconsistências nos projetos dos controles nos painéis de cada avião e, segundo os psicólogos, deveriam ter ocorrido muito mais acidentes do que de fato aconteceram. Essa preocupação com o *lado humano* de um sistema difundiu-se nas mais diversas áreas, inclusive na Ciência da Computação.

Na década de 70, um grupo de pesquisas em *usabilidade de software* da *IBM* reforçou o enfoque que deveria ser dado ao usuário durante o desenvolvimento de sistemas interativos. Vários trabalhos começaram a aparecer, recomendando que usuários típicos fizessem parte da equipe de projeto de software. Esta abordagem ficou conhecida como *projeto participativo.* Em 1986, uma coletânea de artigos [Norman86a] dava um enfoque maior ao *projeto de sistemas centrado no usuário.* Atualmente, diversos periódicos da área de computação têm dedicado números inteiros à divulgação de pesquisa em *HCI.* A partir da década de 80, vários encontros, conferências, *workshops*, têm abordado o assunto. Entre os mais importantes podem ser citados o *UIST (Symposium on User Interface Software and Technology)*, o *CSCW (Conference on Computer-Supported Cooperative Work)*, o *INTERCHI*[2] e o *EWHCI*[3] (*East-West International Conference on Human-Computer Interaction*).

Nos Estados Unidos, o *SIGCHI (Special Interest Group in Computers and Human Interaction)* da *ACM* tinha mais de 6000 membros em 1993. O fato deste grupo estar cres-

[1] disciplina aplicada que visa projetar equipamentos, ambientes, tarefas e ferramentas que se adaptem às necessidades e limitações humanas. Consideram sempre o ponto de vista do usuário; portanto, a ênfase está em como o usuário operará o sistema e não em como o sistema funcionará internamente.

[2] uma combinação do *INTERACT/IFIP* e *CHI/ACM*, foi criada não como mais uma conferência na área, mas como uma cooperação entre as duas principais organizações no campo de *HCI*, agregando, a partir de 1993, a comunidade científica americana e européia. Atualmente é considerada a conferência mais abrangente da área.

[3] O primeiro *EWHCI* aconteceu em 1991, pouco antes do desmembramento da antiga União Soviética. O objetivo dessa conferência é juntar a pesquisa em *HCI* que está sendo desenvolvida nos dois hemisférios.

cendo a cada ano, sendo hoje o maior dos grupos da *ACM*, mostra que a área tem recebido cada vez mais atenção. No Japão, o Ministério da Indústria e Comércio tem criado vários projetos que são conduzidos por consórcios de empresas que investem anualmente milhões de dólares em pesquisa em *HCI*. À medida que os computadores passam a fazer cada vez mais parte do dia a dia, a demanda para monitoramento, gerenciamento, e controle aumenta a necessidade de pesquisa em *HCI*.

2.2 Desenvolvendo Interação Homem-Computador

Interação Homem-Computador é o que acontece quando um usuário humano e um sistema de computação se juntam para executar tarefas. O estudo da interação homem-computador é uma nova e excitante área voltada a responder questões de como melhor fazer esse trabalho de interação. A Figura 2.1, conforme [Hix93], apresenta algumas das diversas sub-áreas relacionadas com o desenvolvimento de sistemas interativos como um todo.

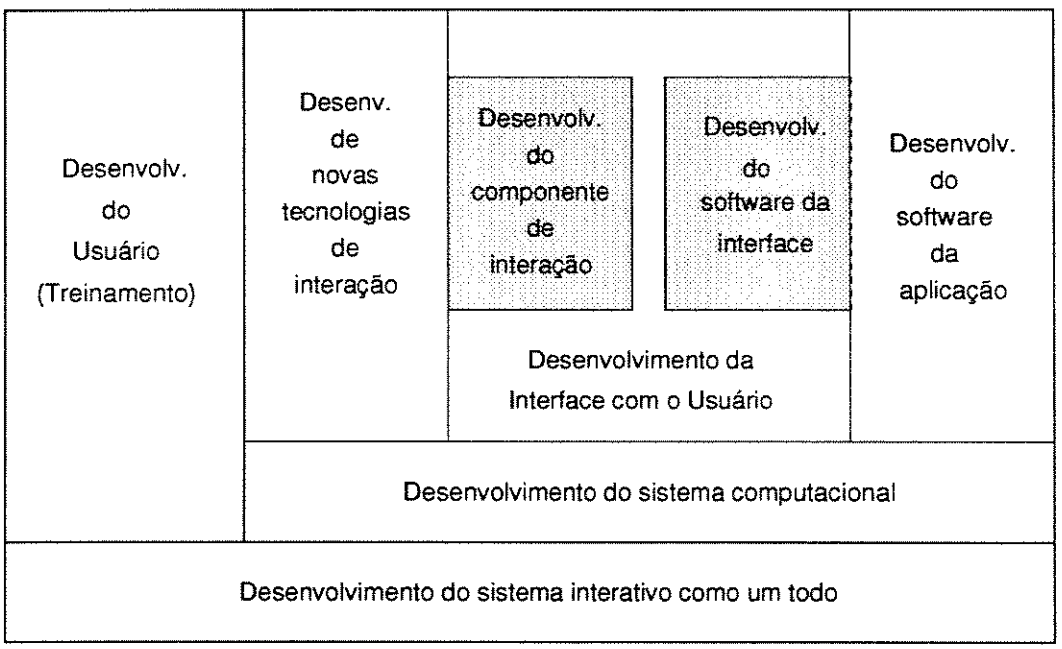

Figura 2.1: *Áreas de desenvolvimento de Sistemas Interativos*

O desenvolvimento da **interface com o usuário** faz parte do processo maior de desenvolvimento de **sistemas interativos**[4], como pode ser observado na Figura 2.1.

[4]Muitas pessoas ainda usam esses termos de maneira errada. Por exemplo, fala-se de projeto de interfaces e projeto de sistemas interativos como se fossem a mesma coisa, ao passo que o último engloba o primeiro.

O **software da aplicação** (software não referente à interface) é desenvolvido com técnicas e metodologias convencionais da Engenharia de Software. Geralmente, existe uma sobreposição não desejável[5] entre o desenvolvimento do **software da interface** e o software da aplicação.

O **conhecimento do usuário** também pode ser aprimorado (principalmente através de treinamentos). Existem vários grupos desenvolvendo pesquisas na área de Educação de Usuários, que envolve principalmente psicólogos, pedagogos e cientistas da computação. Esta também é uma maneira de diminuir a lacuna [Norman86b] existente entre o sistema e o usuário.

O **desenvolvimento de novas tecnologias** tem grande influência nos estilos de interação utilizados no desenvolvimento da interação com o usuário. Muitas dessas novas tecnologias estão relacionadas com o desenvolvimento de hardware e dispositivos que facilitem a interação, como as *touchscreens*, o *mouse 3D* [Venolia93], os teclados *one-handed* [Matias93] e a mesa digital [Wellner93]. Os anos 90 prometem continuar o grande progresso em hardware que irá explorar novas modalidades de comunicação como animação tridimensional em tempo real, sons, imagens e ricos conjuntos de metáforas para interação com o usuário [Marcus91]. Os avanços tecnológicos no desenvolvimento de interfaces também possuem um grande enfoque em software. Com a realização de transmissões em altíssima largura de faixa, na chamada *Digital Data Highway* [Reinhardt94], a transmissão em tempo real de um grande número de imagens se tornará possível. Haverá programas para ler, escrever, arquivar e fazer traduções entre linguagens naturais. Os avanços na tecnologia incluem a construção de interfaces com maior tolerância a erros e ambigüidades na interação entre usuário e computador.

O **componente de interação** está relacionado com a maneira como a interface do usuário funciona, sua aparência e comportamento em resposta ao que o usuário vê, ouve e faz enquanto interage com o computador. O **software da interface** consiste na implementação do componente de interação.

O desenvolvimento do componente de interação de uma interface e o desenvolvimento do software da interface ocorrem em diferentes domínios. Hix e Hartson [Hix93] utilizam os termos *domínio comportamental* e *domínio de construção* para se referir, respectivamente, às áreas de trabalho de pessoas que projetam e desenvolvem componentes de interação de interfaces (*visão do usuário*) e pessoas que projetam e desenvolvem o software relacionado (*visão do sistema*). A compreensão desses dois domínios, o que está envolvido em cada um,

[5]Isso pode gerar uma situação de dependência, onde o programador, para alterar o código da interface, precisa também alterar o código da aplicação.

é muito importante no contexto de especificação de projeto de interfaces, principal assunto tratado neste trabalho.

No *domínio comportamental* são descritas as ações do usuário, percepções e tarefas. A interação é descrita de forma abstrata (isto é, independente do software) em termos do comportamento do usuário e da interface com que ele interage. Desenvolvimento no domínio comportamental envolve fatores humanos, *guidelines* e regras, limitações cognitivas humanas, projeto gráfico, estilos de interação, cenários, especificações de usabilidade, prototipação rápida e avaliação com o usuário. O teste nesse domínio consiste de procedimentos executados pelo usuário.

No *domínio de construção* desenvolve-se o software que implementa o projeto comportamental. São descritas as ações do sistema em relação ao que o usuário faz. Desenvolvimento nessa área envolve *widgets*, algoritmos, programação, bibliotecas de procedimentos, estruturas de dados, controle e fluxo dos dados, diagramas de transição de estados, manipulação de eventos, *callbacks*, representações orientadas a objetos, e alguns tipos de linguagem de descrição de interfaces com o usuário. O teste nesse domínio envolve procedimentos executados pelo sistema.

2.3 Pessoas envolvidas no desenvolvimento de interfaces

Devido ao desenvolvimento de interfaces ser uma área relativamente nova, os papéis a serem desempenhados neste processo, com suas atividades e responsabilidades, ainda não estão bem definidos. Essa é uma área de natureza multidisciplinar que envolve profissionais como psicólogos, *designers* gráficos, escritores, engenheiros de software e programadores. Uma maneira de se estruturar o pessoal envolvido no desenvolvimento de sistemas interativos é apresentada na Figura 2.2 (baseada nas propostas apresentadas em [Bass91] e [Hix93]).

O **projetista do sistema** define a arquitetura global, especificando quais tarefas serão realizadas por que partes do sistema.

O **desenvolvedor da aplicação** cria a estrutura de software necessária para implementar as tarefas realizadas pelo sistema, excluindo a interação homem-computador.

No domínio comportamental, que consiste no lado *humano* do desenvolvimento da interface, o principal papel desempenhado é o do **desenvolvedor da interação com o usuário**, que engloba pessoas com atividades como: definição da classe de usuários, projeto de interação, avaliação de usabilidade e engenharia de fatores humanos. Pessoas nesta área

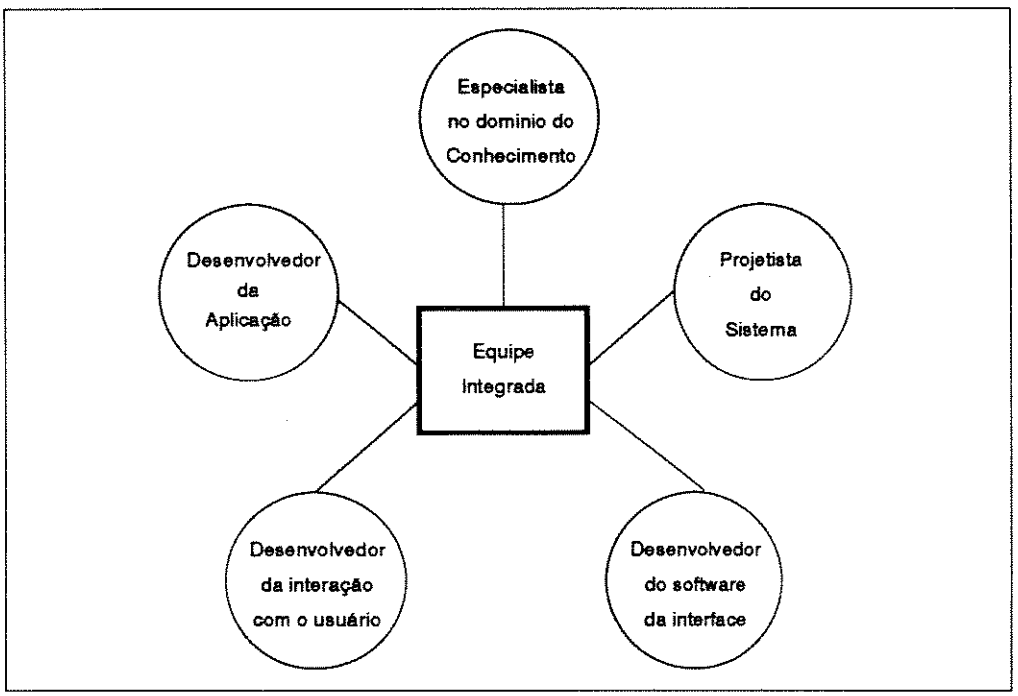

Figura 2.2: *Grupo integrado de desenvolvimento de sistemas interativos*

são diretamente responsáveis pela usabilidade, garantindo a satisfação do usuário. Essas pessoas geralmente estão relacionadas com questões críticas de projeto como funcionalidade da interface, sequenciamento, conteúdo e acesso de informação, formato de menus, operações com uso de mouse e garantia de consistência através da interface. O enfoque principal deste trabalho está nas atividades do desenvolvedor da interação com o usuário, principalmente a especificação de diálogo.

O domínio de construção, que consiste do lado mais *computacional* do desenvolvimento da interface, inclui áreas envolvendo máquinas, ciência da computação e linguagens de programação. O papel mais importante nesse domínio é o do **desenvolvedor do software da interface**. Pessoas neste papel são responsáveis pela tradução do projeto de interação em projeto de software e sua implementação. Elas são responsáveis pela produção da interface com a aparência e comportamento descritos no domínio comportamental.

Um outro importante papel no desenvolvimento de interfaces é o do **especialista do domínio do problema**, que consiste de pessoas que têm conhecimento profundo da área de aplicação do sistema. Geralmente, estes especialistas estão representados pelos futuros usuários ou operadores que irão utilizar o sistema interativo final.

Pessoas com esses diferentes tipos de função devem trabalhar em conjunto e compartilhar o projeto e desenvolvimento, não somente da interface com o usuário, mas do sistema interativo como um todo, como mostrado na Figura 2.2. A cooperação entre as pessoas dessas áreas, com complementação de papéis, é essencial para o desenvolvimento de sistemas de alta qualidade. Essa visão se adapta à, tão em moda, *visão holística do sistema japonês de produção*, que envolve todas as pessoas em todas as fases do desenvolvimento.

2.4 Uma comparação entre desenvolvimento tradicional de software e de interfaces

O desafio de se projetar bons sistemas interativos não passou desapercebido pela Engenharia de Software. Entretanto, tem-se observado que os métodos convencionais utilizados não têm levado a resultados satisfatórios. As metodologias existentes da Engenharia de Software não fornecem guias suficientes para o caso de sistemas interativos [Karat90]. A maioria dos métodos que são utilizados hoje em dia foram desenvolvidos antes das aplicações interativas se tornarem importantes.

As abordagens tradicionais da Engenharia de Software decompõem o sistema hierarquicamente em termos de dados e funções do sistema. A *análise estruturada* [Gane79] (e suas extensões), seguida pelo *projeto estruturado* [Yourdon79], identificam módulos de programa, relegando a tarefa de estabelecer uma interface homem-computador para uma *sub-fase* do desenvolvimento do sistema. A *análise orientada a objetos* [Coad90] mantém o enfoque sobre os mesmos componentes da análise estruturada (dados e funções); entretanto, o enfoque principal está nos dados, e menos ênfase é dada às funções que podem ser executadas sobre os dados. Já a metodologia de desenvolvimento de sistemas de Jackson tradicional (*JSD*) exclui toda a área de engenharia humana[6]. Apesar dessa variedade de abordagens da Engenharia de Software, nenhuma modela a interface com o usuário explicitamente [CarterJr91].

[6] Existe uma proposta de extensão de *JSD*, em [Lim90], que tenta integrar fatores humanos à metodologia.

Todos esses fatos têm levado pesquisadores em todo o mundo a estudar melhor o *processo de desenvolvimento* de sistemas interativos. Metodologias de desenvolvimento de software, notações para especificar o projeto de interfaces e conjuntos de *guidelines de projeto* têm sido propostos, ou reformulados, para se adaptarem a essa nova concepção de desenvolvimento de sistemas – *centrada no usuário.*

Existem várias diferenças e semelhanças entre o desenvolvimento de software de sistemas interativos e o desenvolvimento tradicional de software. Geralmente, muita ênfase é dada ao aumento de *produtividade de software* (com o uso de linguagens de alto nível, abstração, ambientes de desenvolvimento, ferramentas *CASE*[7] e reutilização de software). Qualidade de software geralmente está relacionada à corretude, segurança e manutenibilidade do sistema. Raramente, qualidade de software inclui qualidade da interface com o usuário.

Devido às interfaces com o usuário serem implementadas com o uso de software, muitas pessoas acreditam que as *técnicas bem estabelecidas* para desenvolvimento de software em geral podem ser aplicadas no desenvolvimento de interfaces. Na verdade, os mesmos tipos de conceitos podem ser aplicados aos dois tipos de desenvolvimento – definição de requisitos, especificação, projeto, métricas, avaliação, manutenção, documentação, ciclo de vida – mas para a interface, cada uma dessas atividades deve ser feita de uma maneira diferente, pois neste processo está envolvido um *co-processador de comportamento altamente imprevisível*: o usuário [Hix93].

Uma boa maneira de enxergar o relacionamento entre o desenvolvimento da interface e o desenvolvimento do software da aplicação é analisando como as atividades, em cada tipo de desenvolvimento, estão modularizadas.

2.4.1 Atividades no desenvolvimento tradicional de software

As atividades do desenvolvimento tradicional de software podem ser esquematizadas, de maneira geral, conforme a Figura 2.3 [Hix93]. Um estudo mais abrangente dessas diferentes atividades (fundamentos, documentos produzidos, metodologias) pode ser encontrado em [Pressman92].

O resultado da **análise do sistema** é um conjunto de requisitos para os projetistas de software. Estes requisitos são declarações dos objetivos do sistema, necessidades, funcionalidade desejada, e características nas quais o projeto de software estará baseado.

[7] *Computer-Aided Software Engineering*

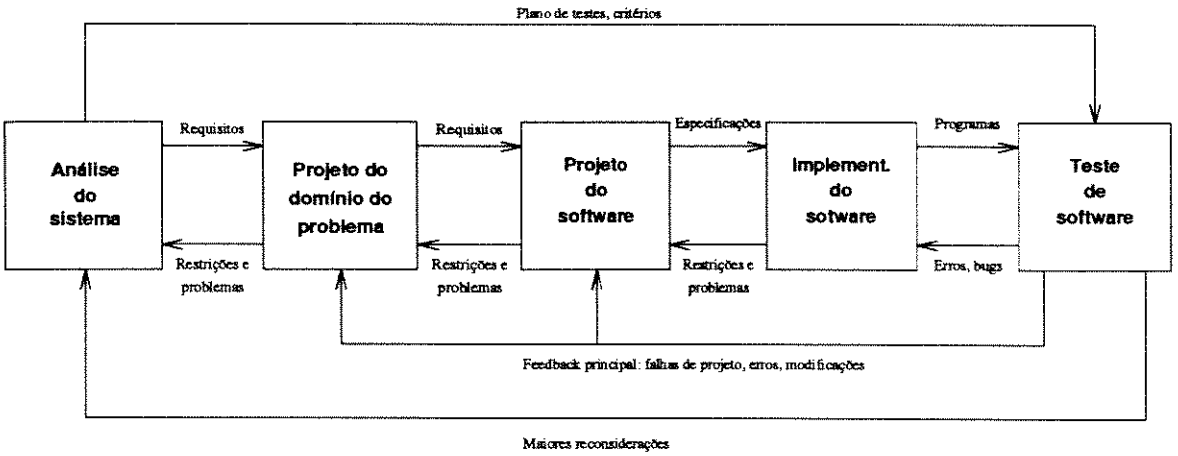

Figura 2.3: *Atividades no desenvolvimento tradicional de software*

Na fase de projeto, os requisitos são traduzidos para representações de software. **O projeto do domínio do problema** consiste de um projeto abstrato, independente do projeto de software e implementação. Entradas para esta atividade são originadas da análise do sistema, onde são decididas quais as necessidades e características do sistema que está sendo projetado. Pode-se associar essa fase ao *projeto preliminar* [Pressman92], relacionado com a transformação dos requisitos em arquitetura de dados e de software.

No domínio do **projeto do software**, o projetista ainda trabalha num nível alto de abstração, sem se preocupar com detalhes de implementação, ao contrário do que acontece no domínio de **implementação**. Em [Pressman92] essa fase está relacionada ao *projeto detalhado*, que consiste em refinamentos da arquitetura do software, produzindo estruturas de dados detalhadas e representações algorítmicas do software.

O teste de software retorna para os projetistas indicativos da presença de defeitos, implicando em modificação nas especificações. Alguns tipos de defeitos também podem ser retornados diretamente para o processo de implementação (por exemplo, *bugs* no código).

2.4.2 Atividades no desenvolvimento de interfaces com o usuário

As atividades envolvidas no desenvolvimento do software da interface podem ser vistas de maneira análoga àquela referente ao desenvolvimento do software da aplicação, como ilustra a Figura 2.4 [Hix93].

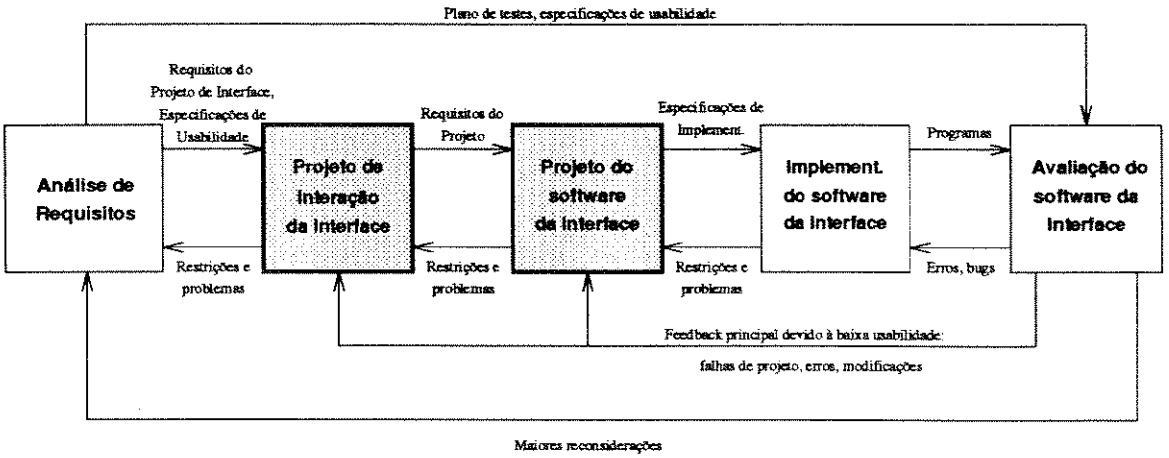

Figura 2.4: *Atividades do desenvolvimento da interface*

As atividades em cada uma das caixas são descritas na Seção 2.5, quando é apresentado um modelo de um ciclo de vida para desenvolvimento de interfaces.

2.4.3 Visão integrada dos processos

A Figura 2.5, conforme [Hix93], mostra a integração das atividades de desenvolvimento referentes à interface e ao software em geral (ou software da aplicação), discutidas anteriormente. É importante notar que a Análise do Sistema e Teste/Avaliação estão sendo compartilhados pelo desenvolvimento do software da interface e pelo software da aplicação.

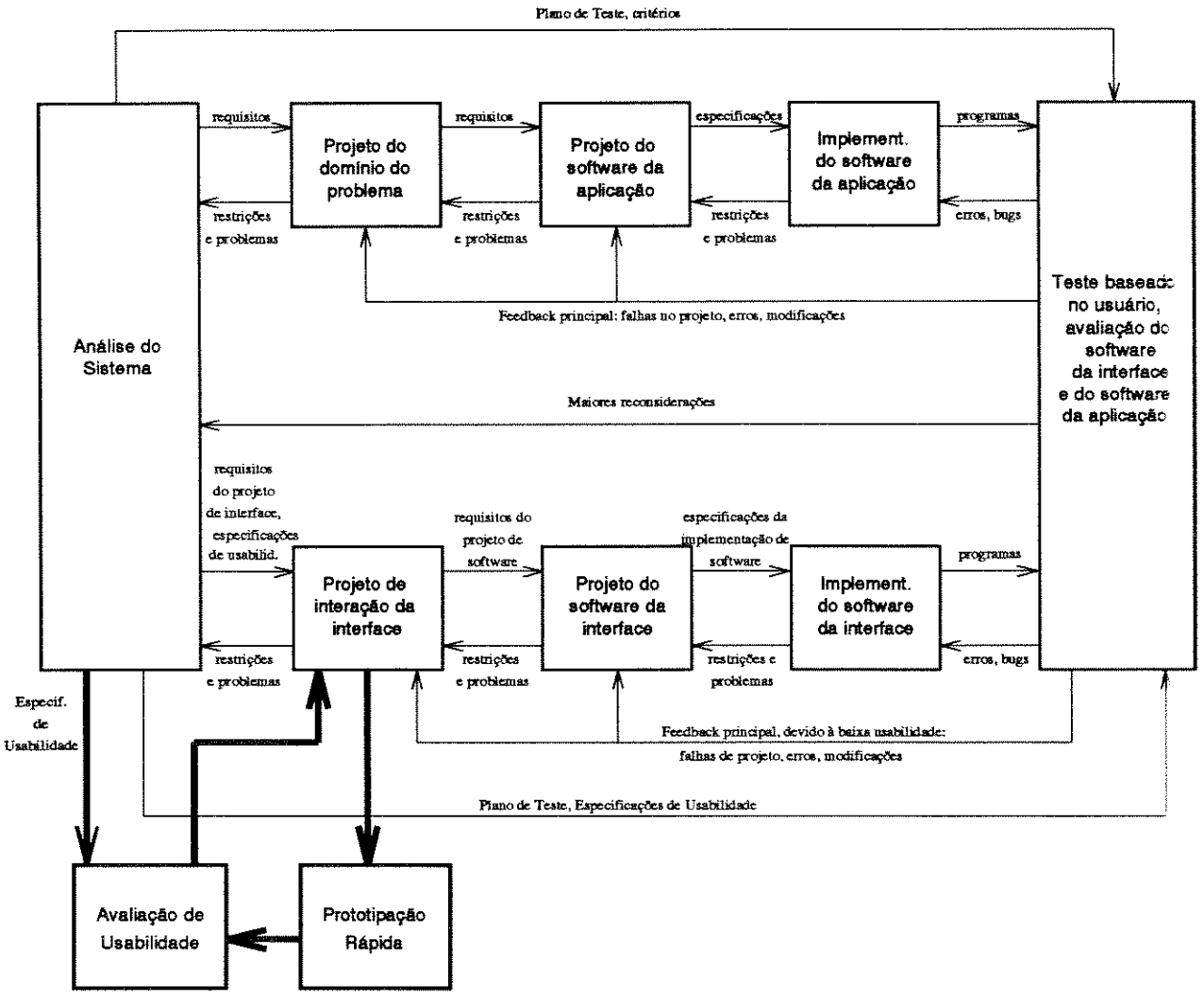

Figura 2.5: *Visão Integrada dos processos de desenvolvimento*

É importante ressaltar que as conexões nos diagramas apresentados nas Figuras 2.3, 2.4 e 2.5 não devem ser vistas como conexões temporais, referentes a algum seqüenciamento de atividades, mas sim como *caminhos de comunicação entre as várias atividades de desenvolvimento.*

Na realidade, existem muitos *canais verticais de comunicação* – entre os desenvolvedores do software da aplicação e do software da interface – conectando as caixas nos dois caminhos apresentados na Figura 2.5. Estes canais de comunicação são essenciais para uma equipe integrada de desenvolvimento. Pela figura também percebe-se a grande distância de comunicação entre o protótipo da interface e o software da aplicação que está sendo cons-

truído. Ainda é uma tarefa difícil o teste da funcionalidade da aplicação com o protótipo da interface, como um protótipo global do sistema.

A necessidade de enxergar o desenvolvimento da interface como parte integral do processo de engenharia de software é fundamental para a obtenção de um sistema de boa qualidade. A interface não pode ser vista como um componente a ser acoplado *a posteriori* ao processo de desenvolvimento da aplicação, nem deve ser desenvolvida isolada do resto da aplicação. A *independência de diálogo*, preconizada por Hartson e Hix, em [Hartson89], consiste na separação, e não no isolamento, do projeto da interface e do projeto da aplicação, de tal forma que alterações na interface não afetem o componente computacional e vice-versa.

Na seção seguinte apresentam-se algumas abordagens comumente utilizadas para o processo de desenvolvimento de software em geral e um modelo específico que enfatiza o desenvolvimento da interface com o usuário.

2.5 Um ciclo de vida para o desenvolvimento de interfaces

Do ponto de vista de software, um sistema passa por várias fases até que seja implementado e colocado disponível; mesmo após a entrega, o sistema ainda pode sofrer modificações. Estas várias fases constituem o que se entende por *ciclo de vida* de um sistema.

Na engenharia de software tradicional os sistemas são geralmente, desenvolvidos utilizando-se uma abordagem *top-down* baseada na decomposição funcional. Um ciclo de vida típico associado a esse tipo de metodologia é conhecido como *Modelo Cascata*[Boehm76] (Figura 2.6).

Neste modelo, cada fase produz resultados que fluem para o próximo estágio; idealmente isto ocorre de forma ordenada e linear. Entretanto, na prática, o desenvolvimento de sistemas complexos (como é a maioria dos sistemas interativos desenvolvidos atualmente) não se adequa à natureza seqüencial do ciclo de vida clássico. A idéia de que esta abordagem não é adequada para o desenvolvimento de sistemas interativos origina-se da natureza *tentativa e erro* do desenvolvimento de interfaces, que consiste de um processo altamente cíclico. Outros problemas apresentados em [Pressman92], relativos a esta abordagem de desenvolvimento estão relacionados com a dificuldade do cliente em apresentar todos os requisitos previamente, dificuldade de acomodar a incerteza natural que existe no início de muitos projetos e o fato do tempo de obtenção de uma versão preliminar do sistema geralmente ser grande.

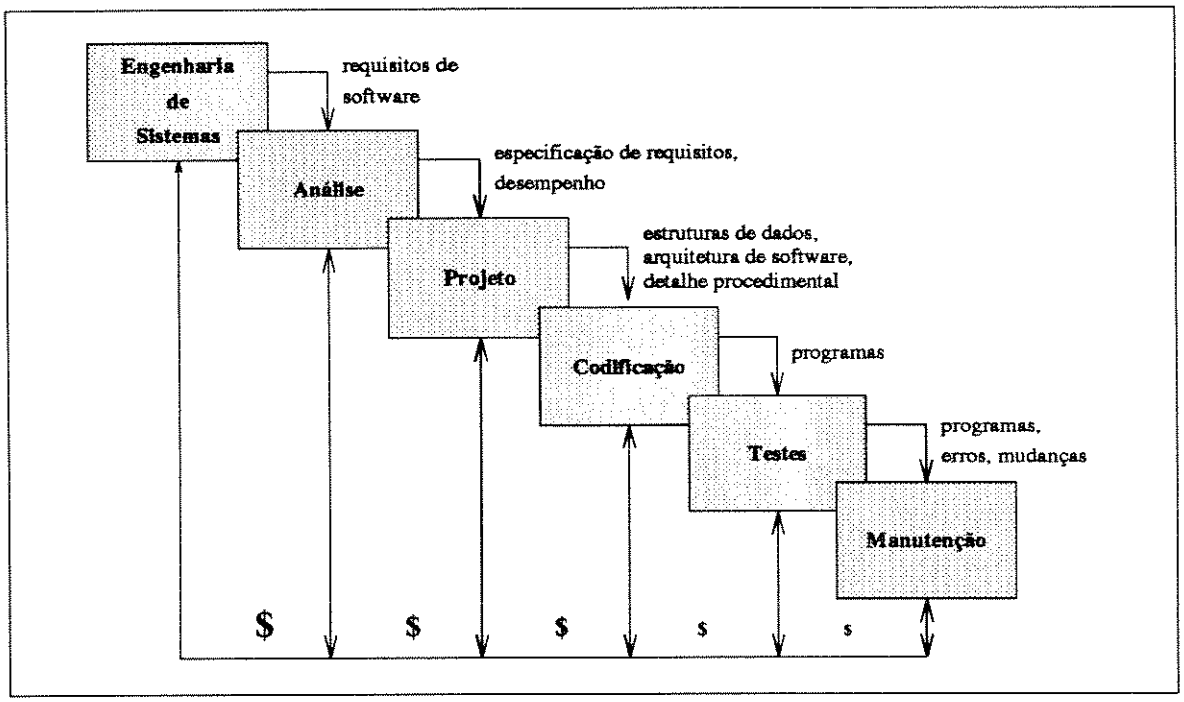

Figura 2.6: *Ciclo de Vida Clássico (Watterfall)*

Boehm, em [Boehm88], reconhecendo a necessidade de uma maior rapidez na alternância entre as várias fases do ciclo de vida, introduziu o *Modelo Espiral* para desenvolvimento de software. Esta abordagem procura combinar as melhores características do ciclo de vida clássico e da prototipação: consiste de um processo que vai incluindo detalhes ao sistema, movendo-se várias vezes através de um processo *top-down*. Trata-se de um *modelo incremental*, pois o protótipo vai sendo melhorado a cada iteração do ciclo (Figura 2.7). Um estudo bem detalhado sobre os modelos cascata e espiral pode ser encontrado em [McDermid91]. Em [James91] é proposta uma adaptação do modelo espiral para o caso de desenvolvimento de interfaces.

No desenvolvimento de sistemas interativos é fundamental que se possa prever o desempenho de uma interface à medida em que ela está sendo construída. *Rapidez e facilidade de modificação* são indispensáveis neste processo; portanto, o modelo espiral não é o mais adequado para o desenvolvimento de interfaces, pois as fases de construção de protótipos e testes ocorrem em ciclos completos de desenvolvimento, como se pode observar na Figura 2.7. Uma abordagem evolutiva, com a construção de protótipos e rápida alternância entre as fases é necessária. *A forma mais confiável de se produzir interfaces de qualidade é através de um processo iterativo que testa protótipos com os usuários e modifica o projeto baseando-se nos comentários recebidos* [Myers89]. Entretanto, o tempo entre construção-avaliação-

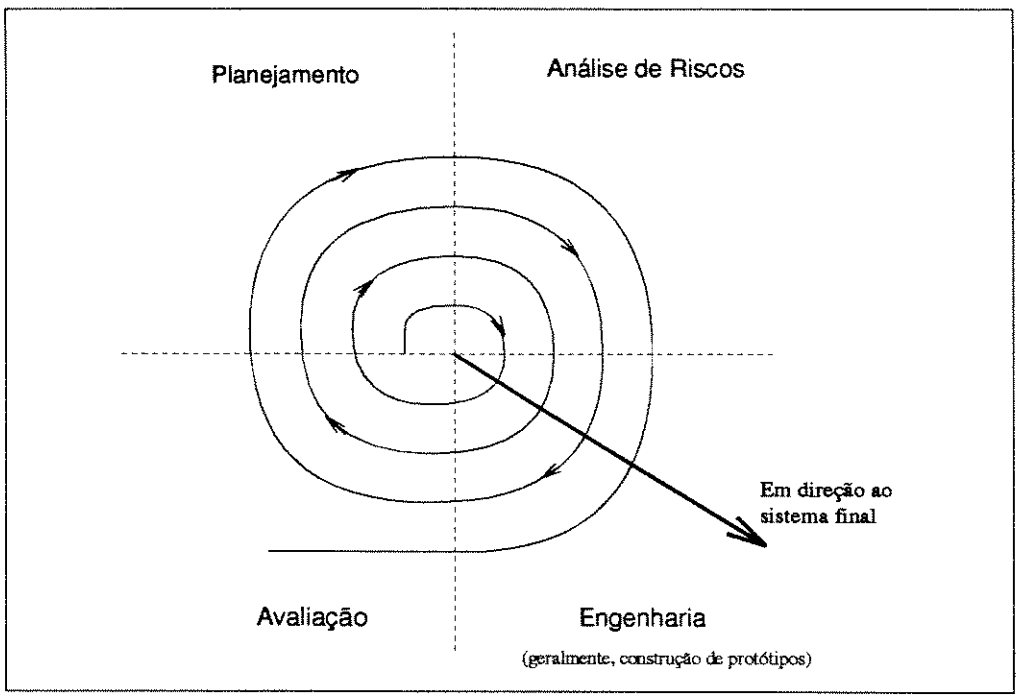

Figura 2.7: *O Modelo Espiral (básico)*

modificação dos protótipos deve ser pequeno. Uma outra crítica geralmente feita ao modelo espiral é que, na maioria das vezes, o protótipo acaba se transformando no sistema final, por não haver uma fase específica de produção do software.

Em [Hartson89] é proposto o *Modelo Estrela* (Figura 2.8). O objetivo deste modelo é fornecer uma estrutura de desenvolvimento mais adequada para interfaces, e é resultado de observações empíricas de pessoas que desenvolveram diferentes tipos de interfaces. Uma das bases do modelo estrela está no que foi chamado de *ondas alternadas*, que se refere ao fato de que o projeto de interfaces não é uma atividade exclusivamente *top-down*, nem *bottom-up*, mas uma mistura das duas. O projetista da interface pode alternar entre elas repetidas vezes durante o desenvolvimento.

Conforme destacado na Figura 2.8 as atividades de maior interesse neste trabalho são: *análise de requisitos* (principalmente a análise de tarefas e funcional) e *projeto* (principalmente as técnicas de representação).

As principais vantagens apresentadas pelo modelo estrela para desenvolvimento da interface com o usuário são:

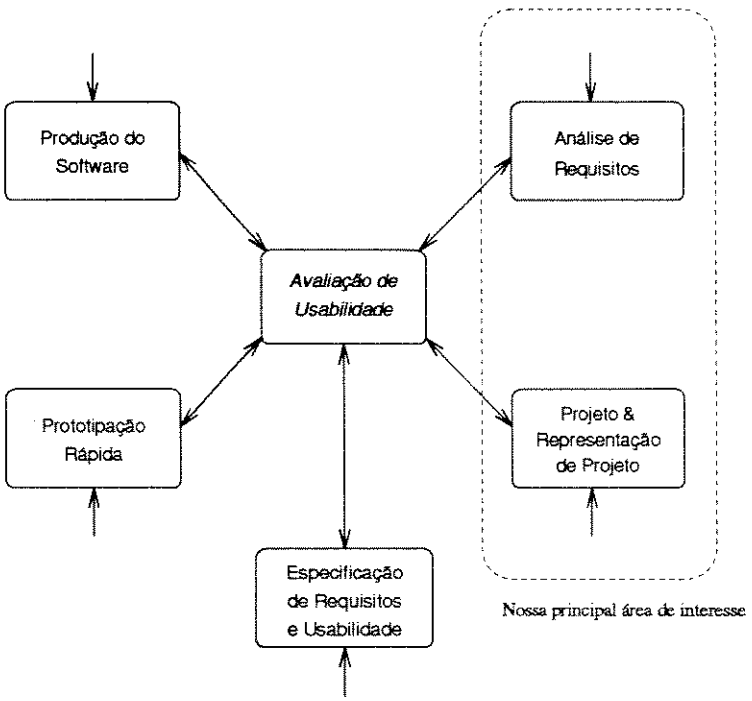

Figura 2.8: *O Modelo Estrela*

- qualquer que seja a atividade atual em desenvolvimento, esta poderá ser rápida e facilmente substituída por outra, existindo uma alta interconectividade entre todas as atividades;
- minimiza o número de restrições entre as diversas fases de um ciclo de desenvolvimento. Por exemplo, não é necessário especificar todos os requisitos antes de trabalhar na fase de projeto;
- este modelo *contempla especificamente o desenvolvimento da interface com o usuário*, diferindo de modelos tradicionais como cascata e espiral, propostos para desenvolvimento de sistemas em geral;
- procura reduzir o tempo entre a construção de protótipos, testes e modificações da interface, pois enfatiza *pequenos ciclos de desenvolvimento*;
- separa explicitamente a fase de produção de protótipos da fase de produção do software propriamente dito; o protótipo deve servir apenas como auxílio para produção do sistema final;
- apresenta uma boa abordagem para o desenvolvimento da interface no domínio comportamental, no qual pretende-se concentrar neste trabalho;

- o modelo estrela está centrado na avaliação de usabilidade da interface, ou seja, o produto de cada atividade é avaliado antes da passagem para uma próxima atividade.

Devido a essas características resolveu-se abordar o processo de desenvolvimento de interfaces (Seção 2.4.2) com o enfoque do modelo estrela. Nas seções seguintes são discutidas as várias fases envolvidas nesse modelo (conforme Figura 2.8).

2.5.1 A atividade de análise

A fase de análise de sistemas interativos não difere muito da atividade já conhecida para desenvolvimento de software em geral. Da mesma forma, o mesmo cuidado e atenção dispensados nesta fase implicam num enorme impacto na qualidade do produto final. É importante que o usuário (*especialista do domínio do problema*) tenha participação efetiva nesta fase. O principal obstáculo à boa comunicação analista/usuário é a *atitude* do analista. Muitas vezes o analista coloca-se numa posição de superioridade por entender de sistemas de computação e um mesmo entendimento da área por parte do usuário é cobrado.

Em [Hix93] são apresentadas as atividades que geralmente estão envolvidas na análise de sistemas interativos:

1. **Análise de necessidades**:

   O objetivo da *análise de necessidades* é estabelecer se o novo sistema realmente é necessário, baseado nos objetivos da organização e nas demandas de mercado. Nesta fase, são determinados os objetivos básicos e as características desejadas do sistema que será construído.

2. **Análise do usuário**:

   Em todo projeto de desenvolvimento, e especialmente para um domínio de problema não familiar, é necessário *entrevistar e examinar usuários* potenciais para determinar que informação será necessária e como ela será apresentada. O resultado é um conjunto de definições de classe de usuários, também conhecido como *perfil de usuários*. As diferenças individuais dos usuários é um dos fatores de maior impacto na usabilidade final do sistema.

   Existem muitas características de usuários que podem ser determinadas, como o conhecimento de computação, o conhecimento no domínio do problema, o hardware com que eles estão acostumados a trabalhar (ex.: *Mac, PC, workstation* e *mainframe*), o

software que é mais familiar (ex.: editores de texto, planilhas e bancos de dados), experiência com aplicações similares ou relacionadas, nível de habilidade (ex.: digitação) e nível intelectual.

3. **Análise de tarefas**:

   A *análise de tarefas* é uma das mais importantes atividades da análise para desenvolvimento de interfaces e procura formalizar e estender o que já foi gerado nas atividades anteriores. Ela fornece uma descrição completa das tarefas, subtarefas e métodos envolvidos no uso do novo sistema, identificando recursos necessários para os usuários e o sistema, de forma cooperativa, executarem estas tarefas. É importante relembrar que as tarefas devem ser expressas a partir da visão que o usuário tem do sistema que está sendo desenvolvido.

   A análise de tarefas geralmente é feita de forma interativa: à medida em que os usuários vão observando e executando as tarefas já estabelecidas no sistema, eles vão sugerindo mudanças e adição de novas tarefas. Estas novas características devem ser adicionadas aos documentos gerados nas fases de análise de necessidades, tarefas, funções e requisitos.

4. **Análise funcional**:

   A Análise funcional consiste de uma descrição das funções do sistema que será projetado e implementado no software que suporta as tarefas identificadas pelo usuário na análise de tarefas.

5. **Alocação de tarefa/função**:

   Devido ao uso de um sistema interativo ser um problema de execução de tarefas cooperativas entre usuário e sistema, deve-se decidir o que será executado por cada uma das partes: para algumas tarefas haverá a participação do usuário e uma função associada do sistema; outras serão alocadas para o sistema (*tarefas automatizadas*); enquanto outras serão alocadas somente para o usuário (*tarefas manuais*).

6. **Análise de requisitos**:

   A *Análise de requisitos* consiste de um processo formal de especificação dos requisitos de projeto para o sistema. É baseada na análise de necessidades, análise de usuários, análise de tarefas e análise funcional.

2.5.2 O projeto da interface

Depois de completada a tarefa de análise e determinadas as várias características do sistema, o próximo passo no desenvolvimento é o *projeto*: considerado como uma das fases mais criativas e mais difíceis do ciclo de vida. O projeto de interfaces envolve uma séria análise de compromissos: uma facilidade a mais dada para o usuário pode implicar numa série de problemas como espaço de memória, frustração de usuários mais experientes e lentidão do sistema. *É difícil escolher a solução ótima, mas a pior pode ser evitada.* O objetivo principal da fase de projeto é chegar a uma versão utilizável do produto, que possa ser implementada.

Dois termos bastante utilizados em *HCI* são *projeto* e *desenvolvimento* de sistemas interativos. O termo *desenvolvimento* refere-se a atividades mais abrangentes na produção de interfaces, incluindo projeto, análise, prototipação, avaliação e modificação iterativa. A atividade de *projeto* consiste da maior atividade no processo de desenvolvimento da interface. Essa atividade é considerada uma tarefa não-trivial, talvez pela *arte* que ainda esteja envolvida neste processo. Em [Hooper86] é feita uma série de analogias bastante interessantes entre projetos na área de arquitetura e projetos na área de interfaces.

O projeto de interfaces tem uma parte relacionada à interação com o usuário e outra relacionada ao software que implementa essa interação. Essas duas atividades distintas são descritas a seguir:

1. **Projeto da interação com o usuário:**
   Esta fase pode ser subdividida em duas subfases: *projeto conceitual* e *projeto detalhado.* O *projeto conceitual*, de nível de abstração mais alto, estabelece a interação básica. O projetista deve começar a identificar possíveis objetos que poderão ser utilizados na interface, decidir como estes objetos serão apresentados ao usuário conceitualmente na interface (sem se preocupar com detalhes de aparência), determinar as operações que serão executadas na interface como resultado da análise de tarefas. O projeto conceitual da interface é às vezes descrito através de *metáforas*[8]. O *projeto detalhado* (ou funcional) especifica a funcionalidade detalhada da interface, que informação é necessária para cada operação em um objeto, que erros podem ocorrer, como esses erros serão tratados, e quais serão os resultados de cada operação. O projeto detalhado também está relacionado com atividades tais como determinar as palavras usadas nas mensagens com o usuário, rótulos e menus, a aparência dos objetos na tela, e o sequenciamento de telas.

[8] maneira de levar o usuário a fazer uma série de analogias com o que é familiar para ele, por exemplo, uma máquina de escrever, uma mesa de escritório.

2. **Projeto do software da interface:**
   Esta fase envolve os mesmos conceitos do projeto de software em geral. Se a implementação futura da interface for feita através de codificação direta, o que está envolvido são algoritmos, estruturas de dados e a estrutura de chamada dos módulos.

Os dois tipos de atividade de projeto (da interação e do software) contribuem para o desenvolvimento da interface mas requerem diferentes habilidades, atitudes, perspectivas, técnicas e ferramentas. O projeto de software, até mesmo do software da interface, é *centrado no sistema*, enquanto o projeto da interação é *centrado no usuário*, focalizando o comportamento do usuário à medida que ele executa tarefas. A comunicação que ocorre entre essas duas atividades de projeto corresponde à comunicação que ocorre entre os domínios comportamental e de construção, e é assunto de representações de projeto de interação, a ser discutido no Capítulo 3.

O tema principal deste trabalho refere-se às técnicas de representação de *projeto de interfaces*. Na Seção 2.6, discute-se um pouco mais sobre essa atividade de projeto, tão importante no processo de desenvolvimento da interface.

2.5.3 Prototipação rápida

A *prototipação rápida* é muito importante no contexto de desenvolvimento de sistemas interativos, principalmente no desenvolvimento da interface com o usuário [Wilson88]. O modelo estrela enfatiza essa importância, apresentando a prototipação rápida como uma fase distinta, separada de todas as outras, principalmente da fase de produção do software da interface. Vários exemplos de incompletude, ambigüidades e inconsistências do projeto são reveladas através dos protótipos, permitindo ao desenvolvedor da interface resolver esses problemas enquanto existe flexibilidade suficiente para se fazer mudanças. O *ciclo de vida estrela* é centrado na avaliação periódica com o usuário. A abordagem de prototipação para desenvolvimento de sistemas interativos envolve a produção de versões simplificadas do sistema, que ilustrem as características principais do sistema final. Embora a prototipação seja muitas vezes associada a ferramentas de software automáticas, é importante enfatizar que ela deve ser vista como uma técnica, não somente como uma ferramenta. Uma técnica pode ser efetiva até mesmo quando utilizada manualmente, especialmente nos primeiros estágios de desenvolvimento. A utilização criativa de protótipos de papel, transparências sobrepostas, cartões recortados, podem ser muito importantes na avaliação inicial do projeto.

2.5.4 Implementação

A função da interface em um sistema interativo é apresentar ao usuário uma máquina abstrata com a qual ele possa interagir. A estruturação dessa máquina abstrata é geralmente feita organizando-a em hierarquia. Desta forma, é possível empregar os princípios da modularização e ocultamento de informação, de forma que cada nível da hierarquia proteja os demais quanto a modificações. Esta estruturação leva à implementação dos níveis da hierarquia no estilo *cliente/servidor* [Scheifler86], sendo o usuário do sistema o cliente de nível mais alto.

Muitas das implementações das interfaces gráficas atuais seguem o paradigma de programação orientada a objetos. Devido à quantidade de rotinas e o grande número de parâmetros que geralmente apresentam, um programador não usa diretamente a programação de um *sistema de janelas*. Ele faz uso de uma camada orientada a objetos, que fornece uma hierarquia de classes predefinidas. Essas classes encapsulam a aparência e o comportamento de objetos como janelas, menus e ícones, reduzindo a quantidade de código gerada pelo programador e assegurando consistência.

Na verdade, o paradigma de orientação a objetos não está relacionado somente à implementação da interface. Conforme Khoshafian [Khoshafian90], o paradigma é explorado de várias maneiras: para o desenvolvimento (projeto e implementação), na apresentação e na integração de interfaces.

2.5.5 Avaliação de Usabilidade

O modelo estrela está centrado na *avaliação formativa*. Este é o tipo de avaliação da interface *enquanto está sendo construída*: nas primeiras fases e continuamente através de todo o processo de desenvolvimento. Num outro tipo de avaliação, chamado de *avaliação somativa*, o projeto é avaliado somente depois de pronto. Em [Monk93] é explorada a *avaliação cooperativa*, que enfatiza a participação de toda a equipe de desenvolvimento na avaliação da interface que está sendo construída.

O projeto e desenvolvimento da interface com o usuário é um processo de refinamentos sucessivos que dura enquanto existir motivação e orçamento para fazer modificações. Um *mecanismo de controle* deve ser usado pelo projetista para decidir quando parar no processo de refinamentos sucessivos. Esse controle é baseado nas especificações quantitativas de usabilidade. A *Avaliação de usabilidade* é usada para comparar a usabilidade atual do projeto de interação com as especificações, à medida que o desenvolvimento prossegue.

Em [Hix93] são apresentadas as atividades envolvidas na especificação e avaliação de usabilidade de interfaces de sistemas interativos. Da forma como é apresentado, o custo para se fazer esse tipo de avaliação pode ser bastante alto, pois envolve manutenção de tabelas de especificação de usabilidade, preenchimento de questionários[9], gravação em vídeo de sessões de uso da interface para observar as reações do usuário, entrevistas, etc. Nesse tipo de avaliação, que foi denominada *avaliação semântica* da interface, o objetivo é verificar a facilidade de aprendizado da interface, o esforço cognitivo do usuário para executar determinada tarefa e o nível de satisfação do usuário. Uma alternativa razoável, porém menos completa, seria fazer uma simples *avaliação sintática* da interface, ou seja, avaliar a interface verificando a aplicação de *guidelines de projeto* para encontrar possíveis inconsistências.

2.6 A atividade de projeto de interfaces

Nesta seção poderiam ser apresentados vários *guidelines de projeto*, estilos de interação e muitos outros aspectos envolvidos no projeto de interfaces. Entretanto, acredita-se que um estudo desse tipo, por si só, não garante uma alta usabilidade na interface. Um entendimento efetivo do espaço do problema pode ser mais proveitoso que receitas e soluções para problemas específicos.

A seguir são apresentados dois modelos considerados bastante importantes no contexto de projeto de interfaces: o *Modelo snowflake*, do espaço de decisões no projeto de interfaces, e o *Modelo Seeheim*, de arquitetura de software para interfaces. Esses modelos apresentam conceitos que são utilizados como base para discussões futuras.

2.6.1 Um modelo do espaço de decisões no projeto de interfaces

Um modelo consiste de uma representação simplificada de um domínio, servindo como meio conciso de comunicação. Hartson e Hix [Hartson89] comentam que 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.

Em [Hix90] e [Casaday91] é apresentado o chamado *modelo snowflake* (Figura 2.9) de espaço de decisões no projeto de interfaces. Neste modelo, as *decisões de projeto de interface* constituem o ponto fundamental; por exemplo, uma decisão em usar um terminal colorido ou

[9] Um tipo de questionário bastante utilizado é o *QUIS - Questionnaire for User Interface Satisfaction*, considerado o mais completo e validado questionário para determinar a usabilidade de uma interface.

monocromático, um menu ao invés de uma linha de comando, uma palavra específica num determinado diálogo. Uma decisão corresponde a uma seleção de um conjunto de alternativas para uma característica do artefato que está sendo construído. Atividades tais como teste ou prototipação são excluídas deste modelo; o modelo inclui decisões, mas não procedimentos para tomar decisões.

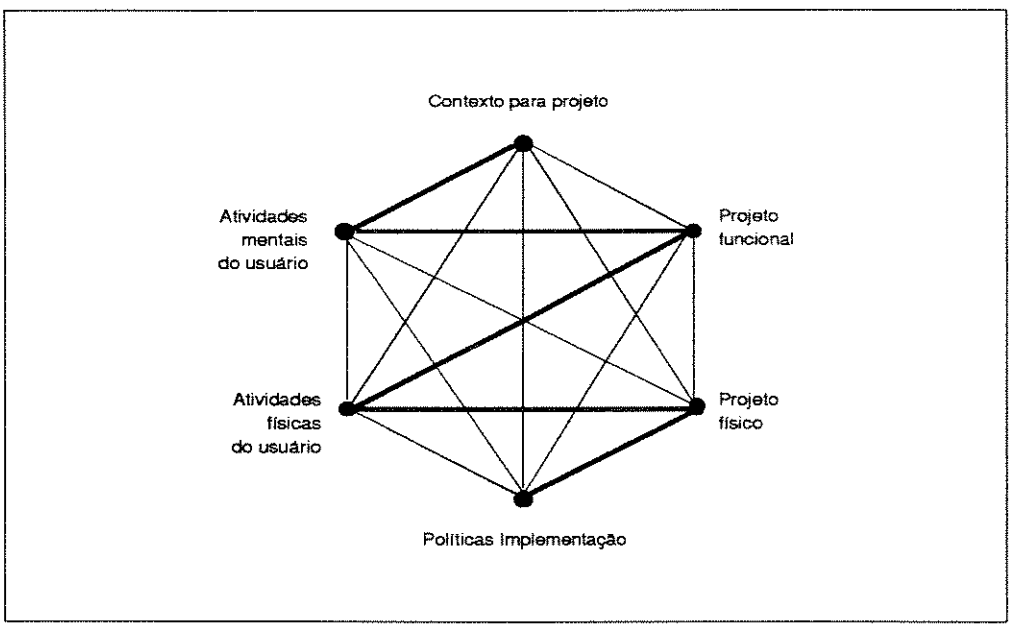

Figura 2.9: *Modelo snowflake do espaço de decisões de projeto de interfaces*

A Figura 2.9 é uma representação gráfica do modelo. Pelo seu formato, a figura sugere *equilíbrio* [Casaday91]: as questões de projeto em cada foco estão relacionadas com as questões de todos os outros focos, não havendo um foco central. O modelo apresenta um bom nível de *granularidade*, pois cada foco pode ser abordado separadamente. Na figura, as linhas em negrito destacam uma possível abordagem linear (*top-down*) para as várias atividades do modelo.

As decisões sobre cada um dos focos devem ser capturadas e representadas à medida que o projeto evolui. O modelo *snowflake* não oferece nenhuma sugestão sobre que técnicas devem ser utilizadas para tal.

Contexto para projeto envolve objetos e eventos externos ao sistema que está sendo projetado; por exemplo, informação de como usuários potenciais estruturam suas atividades, que estratégias eles utilizam, que tarefas são suportadas e quais são ignoradas, tipos de usuários suportados, requisitos de tempo de aprendizado. Este foco inclui, mas não está limitado, aos *objetivos do usuário*, apresentado em [Norman86b], que consiste num conjunto

de estados que o usuário deseja alcançar, utilizando a interface. A coleta de informação para este foco consiste basicamente de entrevistas contextuais e análise tradicional de tarefas. Decisões neste foco são independentes de um sistema específico, sendo melhor evitar referências ao sistema que está sendo projetado.

O foco de **atividades mentais do usuário** envolve o ponto de vista do usuário em relação às tarefas, à funcionalidade de operação do sistema, à maneira como atingir os objetivos utilizando o sistema. Este foco inclui questões relativas ao desempenho cognitivo[10] humano tal como limitações na capacidade de processamento da *memória de curto prazo* e está relacionado com o *modelo do usuário* [Norman86b] referente ao modelo mental que o usuário constrói, resultante da maneira como ele enxerga as ações na futura interface. *Testes com papel e caneta* ou técnicas como *walkthrough cognitivo* [Karat90, Lewis90] podem ser bastante efetivos neste nível. Este foco sugere primeiro projetar a experiência do usuário e então encontrar o sistema que melhor se adequa aos resultados.

O foco de **atividades físicas do usuário** está relacionado com os movimentos físicos do usuário ao interagir com o sistema, em particular, através do hardware. As decisões neste foco geralmente são limitadas, especialmente depois de decidida qual a plataforma de hardware será utilizada. Em [Buxton86] é feita uma crítica aos sistemas de computadores atuais, que fazem uso extremamente pobre do potencial do sistema motor e sensorial humano. A maioria das máquinas utiliza muito mais *sentidos* que o computador, que praticamente só utiliza o tato e a visão.

Projeto funcional envolve decisões semânticas de alto-nível que são independentes de dispositivo. Decisões neste foco produzem descrições abstratas dos objetos da interface com o usuário e as operações a serem realizadas nesses objetos. Esse foco corresponde à *imagem do projeto* [Norman86b], relativo ao modelo conceitual do sistema, construído pelos projetistas, a partir do modelo do usuário. Existem níveis abstratos análogos em muitas outras sub-disciplinas da Engenharia de Software; por exemplo, projeto lógico *versus* projeto físico de banco de dados e algoritmos *versus* programas. Este foco existe porque a lacuna existente entre uma descrição de tarefas e o projeto físico da interface geralmente é muito grande; uma descrição abstrata intermediária é necessária.

Projeto físico envolve decisões sintáticas de baixo-nível, que determinam a aparência e o comportamento do sistema. Decisões neste foco produzem objetos de interface específicos, com aparência, comportamento e posições na tela. Este foco inclui a *imagem do sistema*

[10] O termo *cognitivo* está relacionado com *aprendizado*. A Engenharia cognitiva, aplicada à computação, procura entender os princípios fundamentais atrás de ações humanas no uso de um sistema de computador e, com base nesse estudo, planejar sistemas prazerosos de usar.

[Norman86b], referente ao comportamento da interface gerado pela implementação em software do modelo conceitual do sistema desenvolvido pelos projetistas. As técnicas para representação de projeto de interface, a serem discutidas no próximo capítulo, atuam de forma a tentar reduzir a lacuna que existe entre o modelo do usuário e a imagem do sistema.

O foco de **políticas de implementação** está relacionado com as decisões como plataformas de software e hardware, *toolkits* e cronogramas. Dado um projeto físico apropriado, a implementação pode ser feita de maneira mais direta. Entretanto, as decisões neste foco devem levar em conta quaisquer restrições no projeto que são impostas pelo ambiente de desenvolvimento ou pelo cliente.

2.6.2 Um modelo de arquitetura de software para interfaces

O modelo *snowflake* apresentado consiste em uma abordagem mais geral da grande área de projeto de interfaces, envolvendo ergonomia, psicologia, pedagogia, e ciência da computação. No modelo a ser descrito nesta seção, o projeto de interfaces é abordado com um enfoque mais voltado para a ciência da computação. Este modelo foi resultado de um *workshop* realizado em 1982, na cidade de *Seeheim* na Alemanha. Consiste de um modelo de arquitetura de software para interfaces (Figura 2.10), distinguindo os componentes lógicos que elas devem compreender. O modelo *Seeheim* é bastante geral, sendo assim adequado para descrever várias interfaces existentes. Além disso, a sua descrição é independente de uma interface particular. Isto facilita o estudo do modelo em si, sem influência de uma implementação específica.

O modelo *Seeheim* divide a interface com o usuário em três componentes, como mostra a Figura 2.10.

O **componente de apresentação** envolve a representação física da interface, incluindo dispositivos de entrada e saída, *layout* de tela, técnicas de interação. É a única parte da interface que trata diretamente de dispositivos. As funções do componente de apresentação consistem em capturar dados fornecidos pelo usuário e apresentar na tela dados processados, ou seja, resultados da aplicação. Este componente pode ser enxergado como o *nível léxico* da interface.

O **componente de aplicação** define a comunicação entre a interface com o usuário e o restante do programa (*nível sintático*). Este componente trata da chamada das rotinas da aplicação, e é responsável pelo processamento dos dados fornecidos pelo usuário.

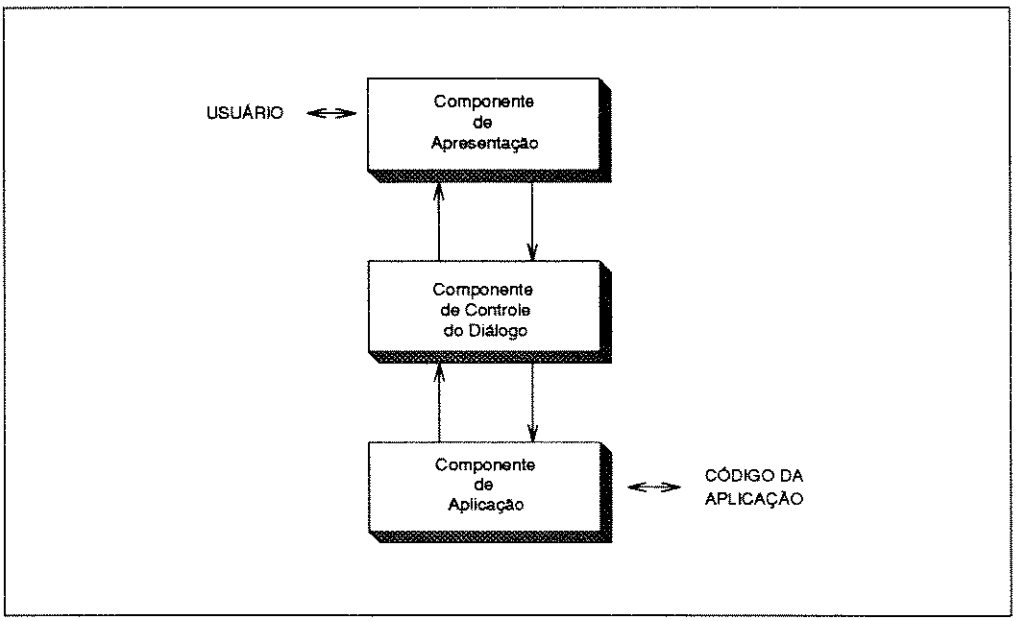

Figura 2.10: *O modelo Seeheim de arquitetura de software para interfaces*

O componente de controle do diálogo trata do diálogo que se dá entre o usuário e o sistema (*nível semântico*). A principal função neste nível é o gerenciamento do fluxo de dados entre os níveis de apresentação e aplicação, ou seja, passar para a aplicação os dados capturados e devolver para o componente de apresentação os resultados do processamento desses dados.

Os três componentes podem ser logicamente vistos como processos separados. Os componentes se comunicam através da passagem de *tokens*. Um *token* consiste na menor unidade de diálogo com significado para uma aplicação. Pode ser um comando, um dado, um parâmetro, opção ou tecla de função. Um *token* fluindo do usuário para o programa de aplicação é chamado de *token de entrada*, e um fluindo da aplicação para o usuário é chamado de *token de saída*.

Essa visão proposta pelo modelo *Seeheim* é bastante utilizada na área de interação homem-computador. Várias ferramentas, metodologias, técnicas para representação da interface, adotam basicamente essa visão da interface em três níveis: léxico, sintático e semântico.

Conforme [Bass91], outras vantagens de se fazer projeto, implementação, teste e manutenção da interface seguindo a arquitetura proposta por esse modelo são apresentadas a seguir:

- facilidade de divisão de tarefas entre os desenvolvedores da interface e os da aplicação;
- facilidade de alterações em um componente sem resultar em alterações nos demais;
- melhoria da consistência da interface;
- exploração da reutilização e portabilidade do software.

2.7 Resumo

Neste capítulo foi apresentada uma visão geral da área de desenvolvimento de sistemas interativos. Tentou-se oferecer uma estrutura para o entendimento dos capítulos seguintes e contextualização do problema a ser tratado. A maior parte do capítulo esteve voltada para a *engenharia de software* relacionada com essa nova área. O texto agrega várias informações que, embora em sua maioria já estejam publicadas, encontram-se dispersas na literatura. Foram apresentadas algumas referências que podem ser consultadas para obtenção de maiores detalhes.

No próximo capítulo é abordado o problema de representação do projeto de interfaces homem-computador. São apresentadas algumas técnicas, em diferentes domínios de representação e para diferentes propósitos.

Capítulo 3

Representação em Projeto de Interfaces

A arte de representar consiste em saber se comunicar.
Sir Ralph Richardson

A obtenção de sistemas interativos com boas interfaces depende da habilidade da equipe entender e avaliar – e conseqüentemente melhorar – o projeto durante o processo de desenvolvimento. Este entendimento depende, em sua maior parte, das técnicas utilizadas para representação do projeto. A necessidade de técnicas efetivas é fundamental no desenvolvimento que enfatiza refinamentos sucessivos e envolve diferentes grupos de trabalho cooperativo para a produção da interface [Hix93]. As diversas pessoas envolvidas possuem requisitos diferentes do sistema e expressam suas necessidades de maneiras variadas [Karat90].

Em [Jacob86a, Jacob86b] é enfatizada a importância da especificação da interface antes de sua construção, pois o projetista poderá assim descrever e estudar uma variedade de interfaces possíveis, sem ter o trabalho de codificá-las primeiramente. Com uma boa técnica de especificação pode-se descrever de forma precisa uma interação homem-computador, livre das restrições relacionadas com a sua implementação. Neste capítulo, são apresentadas algumas técnicas de representação utilizadas para especificar o projeto (da interface) de sistemas interativos. Antes disso é discutida uma possível classificação para essas técnicas, o problema da especificação de interfaces de manipulação direta, diálogos seqüenciais e assíncronos, interfaces modais e não-modais. Apresentam-se algumas das técnicas que aparecem na literatura, enfatizando características principais, comparação com outras técnicas, alguns exemplos, vantagens e problemas na utilização.

3.1 Uma classificação das técnicas de representação

Quando o assunto é representação de projeto de interfaces, algumas das perguntas que geralmente surgem são as seguintes:

- *Quem irá trabalhar com a especificação?*, isto é, *Quem a entenderá e utilizará? Em que domínio ela se encontra?*
- *Qual o principal objetivo da especificação?*
- *A especificação é referente a que?*, ou seja, *O que realmente está sendo especificado?*
- *Quão detalhada deve ser a especificação?*

As classificações para as técnicas de representação de interfaces que geralmente surgem, estão baseadas nessas questões, que estão bastante inter-relacionadas. A seguir, é apresentado um pouco mais do que está envolvido em cada uma dessas questões.

3.1.1 Em que domínio se encontra a especificação?

Várias técnicas têm sido desenvolvidas para descrever interfaces com o usuário. Essas técnicas podem ser divididas em duas grandes classes, dependendo se elas estão mais voltadas para o projeto da interação ou para a implementação da interface [Green86].

Muitas das técnicas atualmente utilizadas para desenvolvimento de interfaces (por exemplo, diagramas de transição de estados, mecanismos baseados em eventos, orientação a objetos) fazem parte do *domínio de construção*, ou seja, a descrição da interface baseada no ponto de vista do sistema. Quem vai entender e utilizar a especificação provavelmente serão os projetistas do software e os programadores da interface.

Como foi apresentado no Capítulo 2, no *domínio comportamental* o desenvolvedor da interação procura especificar a interface do ponto de vista do usuário. Conseqüentemente, as técnicas utilizadas são mais centradas no usuário e *orientadas a tarefas*, que é a maneira como o usuário enxerga a interface.

Muitas vezes, os projetos de interface representados no domínio comportamental precisam ser traduzidos (manual ou automaticamente) para o domínio de construção, para que possam ser implementados; mas a preocupação em descrever primeiramente a interface no domínio comportamental provavelmente irá levar a um sistema final mais amigável para o

usuário. É importante ressaltar que as técnicas de representação no domínio comportamental não pretendem substituir as técnicas do domínio de construção; *cada uma dá suporte a um domínio diferente* [Hix93].

3.1.2 Qual o propósito da especificação?

Uma outra maneira comumente utilizada para classificar as técnicas de representação de interfaces, está relacionada com *qual o objetivo principal da especificação que está sendo feita.* Com essa visão, as técnicas existentes poderiam se encaixar em dois modelos: *de análise* ou *de síntese.*

Os *modelos de análise* representam os processos cognitivos e o conhecimento de tarefas pelo usuário de maneira que o projetista possa estimar vários aspectos da usabilidade da interface, tais como erros de usuário que provavelmente irão ocorrer [Gugerty93]. Os modelos de análise podem ajudar a prever a qualidade da interface, prevendo o desempenho e satisfação do usuário, numa etapa anterior à fase de teste de usabilidade. Grande parte da avaliação da interface é feita com técnicas de simulação, ao invés de usuários reais testando o sistema. Alguns tipos de avaliação analítica utilizam métricas de previsão de desempenho do usuário, que prevêem tempos de execução de ações do usuário tais como movimentos do mouse (ex.: Lei de Fitts [Fitts54]) e teclas digitadas (ex.: *modelo keystroke* [Card83]). Ações cognitivas, perceptuais e da memória podem também ter tempos de execução associados. Outros tipos de avaliação analítica prevêem desempenho indiretamente através da identificação de inconsistências e ambigüidades na interface, determinando equivalência de tarefas, e analisando o fluxo de informação entre tarefas [Reisner81].

Os *modelos de síntese* procuram ser mais simples que os de análise. O principal objetivo é simplesmente descrever ações do usuário numa interface. Os documentos gerados por esses modelos são fundamentais para toda a equipe de projeto, principalmente para os ingressantes, que ainda não estão familiarizados com o sistema em desenvolvimento. Esses modelos podem possuir características que ajudem na avaliação da interface, embora esse não seja o principal objetivo.

3.1.3 O que realmente está sendo especificado?

No campo da interação homem-computador as representações de projeto podem enfatizar as *características cognitivas* da interface, ou as *características funcionais* do sistema. Essas duas características estão bastante interligadas, uma vez que toda função no sistema

está relacionada com uma ou várias ações do usuário. Algumas representações estão mais voltadas para a especificação de ações do usuário, aspectos cognitivos envolvidos, capacidade de lembrança, enquanto outras estão mais voltadas para a especificação da funcionalidade do sistema como um todo.

Já em outras técnicas de especificação, o principal enfoque é a *aparência da interface* (especificação sintática), ou seja, objetos utilizados, posicionamento desses objetos, *layout* das telas e disposição dos menus.

Neste trabalho o principal enfoque está na descrição do diálogo entre o usuário e o computador, e não na especificação de detalhes da aparência da interface.

3.1.4 Quão detalhada é a especificação?

Como é apresentado nas seções seguintes, as técnicas para especificação de interfaces podem apresentar níveis de detalhe tão profundos quanto se desejar. Isto irá depender de fatores como: tipo de interface que está sendo desenvolvida, disponibilidade de especificações de projetos anteriores (*reutilização de especificações*) e tempo e recursos disponíveis.

3.2 Especificação de interfaces de manipulação direta

Uma *interface de manipulação direta* apresenta um conjunto de representações visuais do mundo real na tela, e um conjunto de manipulações que podem ser executadas sobre esses objetos. Através do apontamento de representações visuais dos objetos e ações, o usuário pode executar as tarefas de forma mais rápida e observar os resultados imediatamente [Shneiderman92]. Segundo Laurel, [Laurel86], a manipulação direta aumenta o engajamento do usuário, ou *experiência de primeira pessoa*[1] na interação homem-computador. Ao invés de usar linguagens de comando para descrever operações em objetos que geralmente são invisíveis, o usuário *manipula* objetos gráficos visíveis na tela.

Segundo Schneiderman [Shneiderman83], as principais vantagens na utilização de um sistema com interface por manipulação direta são:

[1] O usuário atua diretamente com participação ativa na interação, por exemplo, a movimentação de arquivos no *Openwindows*. Ao contrário, na *experiência de segunda pessoa*, existe um intermediário entre a interface e o usuário, que faz declarações imperativas ao sistema, por exemplo, copiar arquivos usando linha de comando.

- usuários novatos podem aprender rapidamente, normalmente com uma demonstração feita por um usuário mais experiente;
- usuários especialistas podem trabalhar extremamente rápido;
- mensagens de erro são raramente necessárias, pois os resultados são imediatamente visíveis, e alguns tipos de erro são impossíveis de acontecer;
- usuários podem perceber imediatamente se suas ações levarão aos objetivos pré-estabelecidos e, se não, simplesmente mudam a direção da atividade;
- usuários diminuem ansiedade no uso de sistemas interativos, pois suas ações são mais facilmente reversíveis.

Essa facilidade de interação oferecida pelas interfaces de manipulação direta implica uma maior dificuldade de especificação e programação (*infelizmente, facilidade para o usuário geralmente implica dificuldade para projetistas e programadores*). O principal desafio hoje em dia é especificar justamente esses tipos de interface.

3.2.1 Diálogo seqüencial e Diálogo assíncrono

Dois tipos básicos de diálogo são definidos no campo de interação homem-computador, diálogo seqüencial e diálogo assíncrono. No *diálogo seqüencial*, as tarefas são apresentadas ao usuário, uma por vez, de tal forma que uma segunda tarefa só pode iniciar quando a primeira tiver sido concluída. No *diálogo assíncrono*, ou *diálogo multithread* o usuário pode mudar de uma tarefa (*thread*) para outra em qualquer parte da interação, sem esperar a conclusão da tarefa atual. Este tipo de diálogo é o que comumente ocorre nas interfaces de *sistemas baseados em janelas*.

Diálogo seqüencial e assíncrono apresentam diversas diferenças. As principais são: dispositivos de entradas de dados[2], técnicas para representação do projeto[3] e ferramentas para prototipação.

3.2.2 Interfaces modais e Interfaces não-modais

A maioria das interfaces tradicionais são altamente modais, o que leva os projetista a especificá-las convenientemente com diagramas de transição de estados (Seção 3.4.3). *Modos*

[2]O teclado normalmente é associado a diálogo seqüencial, enquanto o mouse é associado a diálogo assíncrono.

[3]Interações assíncronas são mais difíceis de representar que interações estritamente seqüenciais.

ou *estados* se referem à variedade de interpretações de uma mesma operação de entrada fornecida pelo usuário. Uma *interface modal* requer que o usuário relembre (ou o sistema indique) qual o estado atual da interface e quais os diferentes comandos que se aplicam àquele estado. *Interfaces não modais* não requerem isso; o sistema encontra-se sempre no mesmo modo, e as entradas sempre possuem a mesma interpretação. Em [Thimbleby90] é apresentado um estudo detalhado sobre *modos* em sistemas interativos.

À primeira vista, as interfaces de manipulação direta parecem ser não-modais; vários objetos visíveis na tela, e vários comandos disponíveis para serem aplicados a qualquer hora sobre qualquer um dos objetos. Assim, o sistema parece estar, quase sempre, nesse mesmo *estado universal.* Esta é uma idéia errada, pois, segundo [Jacob86a], as interfaces de manipulação direta são altamente modais. Entretanto, elas são consideradas mais fáceis de usar que as interfaces modais tradicionais devido à maneira direta na qual os modos são apresentados e manipulados.

3.3 Propriedades de uma boa técnica de especificação

O primeiro passo para se fazer um projeto de interface é ter em mãos uma boa notação para armazenar e discutir possibilidades alternativas. A linguagem padrão comumente utilizada para especificação, em qualquer área, é a linguagem natural utilizada pelo projetista (por exemplo, Inglês ou Português). Entretanto, *especificações por linguagem natural*, também conhecidas como *especificações prosaicas* tendem a ser volumosas, vagas, ambíguas, difíceis de provar a corretude, consistência ou completude [Shneiderman92].

Em [Jacob86b] é apresentada uma lista de propriedades que devem ser procuradas na seleção de uma boa técnica para especificação de interfaces:

- a especificação da interface deve ser fácil de entender, ou seja, a interface da técnica de especificação deve ser amigável;
- o esforço para se produzir a especificação deve ser menor que o esforço para se produzir o software que implementa a interface;
- a especificação deve ser precisa. Não podem existir ambigüidades ou dúvidas acerca do comportamento do sistema para cada entrada possível do usuário. *Linguagens formais* e *semiformais* têm mostrado ser efetivas em diversas áreas como Matemática, Física, Projeto de Circuitos, Música e Ciência da Computação;
- a especificação deve facilitar a verificação de consistência na interface;

- a especificação deve ser poderosa o suficiente para expressar sistemas de comportamento não trivial com um mínimo de complexidade;
- deve separar claramente *o que* o sistema faz de *como* ele faz (implementação);
- deve ser possível construir um protótipo do sistema diretamente a partir da especificação da interface. Uma técnica de representação ideal é aquela que, a partir de especificações da apresentação e controle da interface numa linguagem de alto nível, produz como saída o código que implementa a interface especificada;
- a estrutura da especificação deve estar relacionada ao modelo mental do usuário do sistema. Os termos utilizados na especificação devem representar conceitos que sejam significativos para o usuário.

Em [Green86] é comentado que uma notação para projeto pode ser medida de duas maneiras: pelo seu *poder descritivo* (conjunto de interfaces que podem ser descritas pela notação) e pelo *poder de usabilidade* (conjunto de interfaces que podem ser facilmente descritas pela notação). O poder de usabilidade de uma notação consiste num subconjunto do poder descritivo da mesma.

Infelizmente, nenhuma das técnicas atualmente existentes possuem todas as propriedades citadas anteriormente, mesmo porque elas são conflitantes entre si; por exemplo, se a geração automática do código da interface é contemplada pela ferramenta que implementa a técnica, provavelmente os resultados da especificação não serão de fácil entendimento por toda a equipe que está desenvolvendo a interface. Nenhuma notação é apresentada como a solução para os problemas de especificação de interfaces, bem como ainda não existem regras de ouro para a aplicação desta ou daquela notação.

3.4 Algumas técnicas de especificação

Nesta seção descrevem-se algumas técnicas de especificação conhecidas e comumente utilizadas. Não foi firmado como objetivo apresentar um estudo detalhado sobre cada uma dessas técnicas; ao contrário procurou-se apresentar de forma sucinta, enfocando as principais características, possíveis comentários comparando uma técnica com outra, alguns exemplos, vantagens e problemas com as suas aplicações.

As técnicas apresentadas nesta seção são comumente vistas como orientadas ao sistema (domínio de construção) e baseadas nos modelos de síntese. É importante ressaltar

que, apesar de terem essas características, nada impede que elas sejam utilizadas para descrição das tarefas do ponto de vista do usuário ou para avaliação de usabilidade (muitas delas oferecem subsídios para isto).

3.4.1 Gramáticas

As linguagens formais possuem gramáticas associadas. A motivação básica para este modelo é que a interação homem-computador consiste em um diálogo, como na comunicação entre duas pessoas. O projetista pode especificar a sintaxe da interface usando a notação *BNF* (*Backus Naur Form*).

A descrição formal baseada em *BNF* constitui-se de:

- um conjunto de *símbolos terminais* (as palavras da linguagem), associadas às ações primitivas do usuário na interface;
- conjunto de *símbolos não terminais* (algumas sentenças da linguagem);
- *símbolo inicial*;
- *metasímbolos*, por exemplo, "+" (*and*), "|" (*or*), "::=" (*é composto de*);
- *regras*, associadas às ações do usuário na interface.

A figura 3.1 ilustra uma descrição de interface através de uma gramática:

<command> ::= <create> | <polyline> | <delete> | <move> | STOP
<create> ::= CREATE + <type> + <position>
<type> ::= SQUARE | TRIANGLE
<position> ::= NUMBER + NUMBER
<polyline> ::= POLYLINE + <vertex_list> + END_POLY
<vertex_list> ::= <position> | <vertex_list> + <position>
<delete> ::= DELETE + OBJECT_ID
<move> ::= MOVEA + OBJECT_ID + <position>

Figura 3.1: *Um exemplo de especificação por gramática (BNF)*

Reisner [Reisner81] apresenta um exemplo de como *BNF* pode ser usado para descrever uma interface com usuário. Ela utiliza propriedades formais de especificações *BNF* de dois sistemas para prever diferenças no desempenho dos usuários. Segundo Reisner, três aspectos do formalismo podem ser usados para comparar alternativas de projeto:

- *número de símbolos terminais* (número de palavras, ou seja, número total de ações primitivas na interface);
- *número de símbolos não-terminais*, associado com a complexidade do sistema;
- *tamanho das palavras*, ou seja, o número de passos que o usuário deve executar para realizar determinada tarefa;
- *número de diferentes tipos de sentenças* para realizar tarefas similares (avaliação da consistência).

As gramáticas *multiparty* [Shneiderman92] constituem uma extensão das gramáticas convencionais. Através da representação do sistema como uma das partes da interação, essas gramáticas permitem associação direta das entradas fornecidas pelo usuário com o feedback da interface.

As representações baseadas em gramáticas são muito boas para representar interfaces baseadas em linguagens de comando textuais, mas são impróprias para especificar certos estilos de interfaces, como manipulação direta. A maior dificuldade reside na interpretação humana dessas descrições: não-terminais em uma expressão podem levar a outros não-terminais, num processo de sucessivas iterações, muitas vezes recursivas. Dessa forma, é difícil visualizar sentenças numa linguagem através da análise da descrição em *BNF*.

3.4.2 Árvore de menus

Para muitas aplicações, uma *árvore de menus* é um excelente estilo de seleção, como na interface do *Lotus 1-2-3*. Árvores de menus são bastante poderosas como técnicas de especificação de interfaces, pois apresentam de forma sucinta aos usuários, projetistas, programadores e outras pessoas da equipe, a estrutura completa e detalhada do sistema. Uma árvore de menus apresenta relações de alto nível e detalhes de baixo nível. O importante é apresentar a estrutura de menus por inteiro (ver Figura 3.2), de uma só vez, facilitando a checagem de consistência, completude, ambigüidade e redundância.

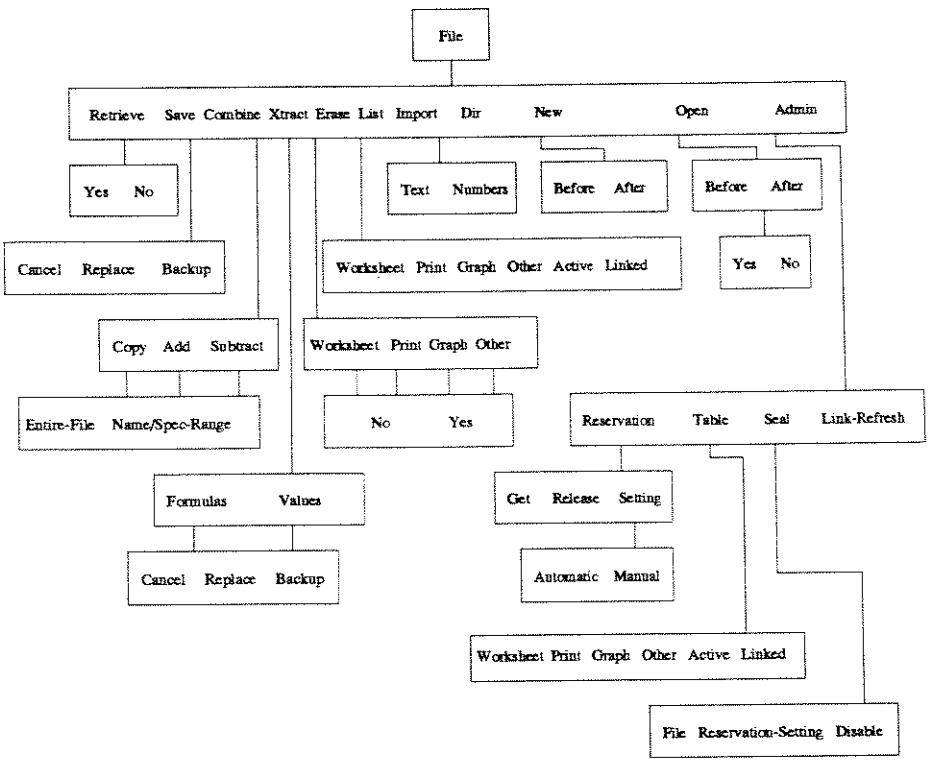

Figura 3.2: *Árvore de menus da opção File do Lotus 1-2-3*

Especificações através de árvores de menus geram basicamente diálogos seqüenciais. Além disso, são consideradas incompletas devido a não apresentarem a estrutura completa de ações possíveis do usuário tais como retornos a menus anteriores ou ao menu inicial, desvios para manipulação de erros ou telas de auxílio ao usuário (*helps*). Entretanto, adicionar todas essas transições pode deixar bastante confusa a estrutura da árvore de menus [Shneiderman92].

Para algumas interações não baseadas estritamente em menus, existe um conjunto possível de estados e transições possíveis entre eles, que podem não formar especificamente uma estrutura de árvore. Para esta e outras circunstâncias, uma notação mais geral, conhecida como *Diagramas de Transição de Estados*, ou *DTEs*, é mais utilizada.

3.4.3 Diagramas de Transição de Estados

Devido a muitas tarefas realizadas pela interface envolverem o tratamento de uma seqüência de eventos de entrada, é natural pensar na utilização de uma rede de transição de estados para especificar a interface. Enquanto as descrições em *BNF* são puramente textuais, os *DTEs* são representações gráficas do fluxo de controle do diálogo seqüencial. Os *nodos*

representam estados da interface ou telas; os *arcos* representam as transições entre estados e são rotulados com o símbolo de entrada que irá provocar a transição para o próximo estado (ver Figura 3.3). Em algumas especificações, arcos podem ser rotulados com rotinas da aplicação a serem chamadas e saídas de dados a serem exibidas a fim de gerar resposta semântica para o usuário do sistema.

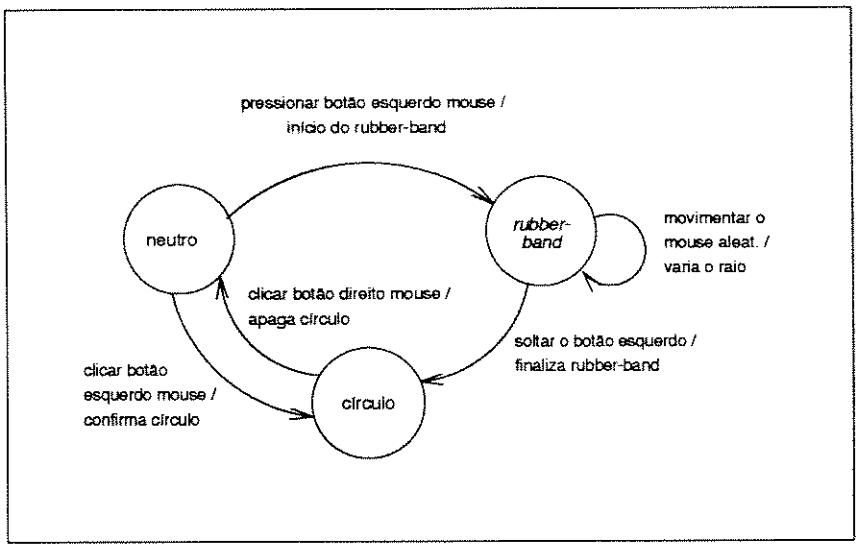

Figura 3.3: *Um exemplo de especificação de diálogo usando DTE*

As notações baseadas em *DTEs* apresentam de maneira explícita o conceito de estado, regras de transição associadas, seqüencialidade de ações, enquanto isso está implícito nas notações baseadas em *BNF*.

Algumas das dificuldades encontradas com os diagramas de transição de estados convencionais são apresentadas em[Lucena92]:

- não fornecem noção de hierarquia de ou modularidade. Em conseqüência, não suportam o desenvolvimento *top-down* ou *bottom-up*;
- um evento que provoca a mesma transição de um grande número de estados deve ser explicitamente especificado para cada estado. Não há como fatorar transições semelhantes (ocasionadas por um mesmo evento) de um grupo de estados. Todos os estados devem ter arcos para todas as entradas que provocam erros e todos os comandos universais como *help* e *undo*;
- o diagrama pode se tornar muito confuso quando utilizado para representar interfaces de sistemas mais complexos, transformando-se num *spagetthi* de transição de estados.

Na verdade, a especificação de interfaces utilizando tanto *BNF* quanto *DTEs* para sistemas não triviais não é tarefa fácil;

- são intrinsecamente seqüenciais, não fornecem uma representação natural para a expressar intercalação e paralelismo entre tarefas.

Reconhecendo alguns desses problemas, são propostas na literatura algumas extensões para os *DTEs* convencionais. Modularidade e hierarquia são possíveis se nodos são incluídos como subgrafos [Shneiderman92]. Dessa forma, o diálogo pode ser decomposto em vários subdiálogos. Em [Green86, Foley90] é ressaltado que o uso de subdiagramas não aumenta o poder descritivo dos *DTEs*, exceto quando os subdiagramas puderem ativar-se recursivamente (*redes de transição recursivas*).Também para suportar interações sensíveis ao contexto, nodos podem conter registradores e funções *booleanas*, que determinam condições para que o nodo possa ser atravessado (*redes de transição melhoradas*).

Em [Jacob86a] é apresentada uma proposta de especificação baseada nas observações de que as interfaces de manipulação direta (onde predomina o diálogo assíncrono) possuem uma estrutura de co-rotinas[4] e podem ser vistas como altamente modais (Seção 3.2.2). Esta especificação combina *DTEs*, para representar o estado de cada co-rotina, com uma forma de linguagem de eventos (Seção 3.4.5), para descrição dessas co-rotinas. Cada *locus* de diálogo é descrito como um objeto separado com um *DTE* único. Os objetos são combinados para definir a interface global como um conjunto de co-rotinas.

3.4.4 Estadogramas

Embora os diagramas de transição de estados sejam efetivos para seguir o fluxo de ações numa interface, eles podem facilmente se tornar grandes e confusos. Esta confusão geralmente acontece quando cada nodo deve apresentar transições para estados de *help*, estado inicial ou estados finais. Concorrência também é pobremente representada por *DTEs*, embora combinações com *redes de petri* possam ajudar [Shneiderman92, Carolis94].

Os *estadogramas*[5] são diagramas propostos por Harel [Harel87], para descrição do comportamento de *sistemas reativos*. Estendem os diagramas de transição de estados, eliminando alguns inconvenientes e retendo a natureza gráfica dos mesmos, facilitando a compreensão. Outra característica importante é que os estadogramas são formais; em conseqüência,

[4] consiste em uma variação de uma rotina comum. Quando uma rotina chama a outra, a execução não se inicia no ponto de entrada, mas no último ponto executado da rotina chamada.

[5] neologismo da palavra *statechart*, apresentado em [Figueiredo91].

permitem manipulação, manutenção e análise automática. As extensões acrescentam concorrência, comunicação (*broadcast*) e hierarquia. Harel sugere várias aplicações para os estadogramas: especificação de diálogo de interfaces, descrição de hardware, protocolos de comunicação, sistemas de tempo real, entre outras.

Da mesma forma que os diagramas de transição de estados, estadogramas são baseados em ações, transições, estados, eventos e condições. Eventos e condições podem ser combinados para causar transições entre estados. Ações são as respostas dos estadogramas às transições. Portanto, a entrada correspondente a uma especificação em estadograma compreende estímulos (eventos) externos e internos, e a saída corresponde às ações.

Em [Lucena92][6] é apresentado um exemplo comparativo entre estadogramas e *DTEs*, onde é ressaltada a superioridade de clareza e concisão na notação baseada em estadogramas, como pode ser observado na Figura 3.4.

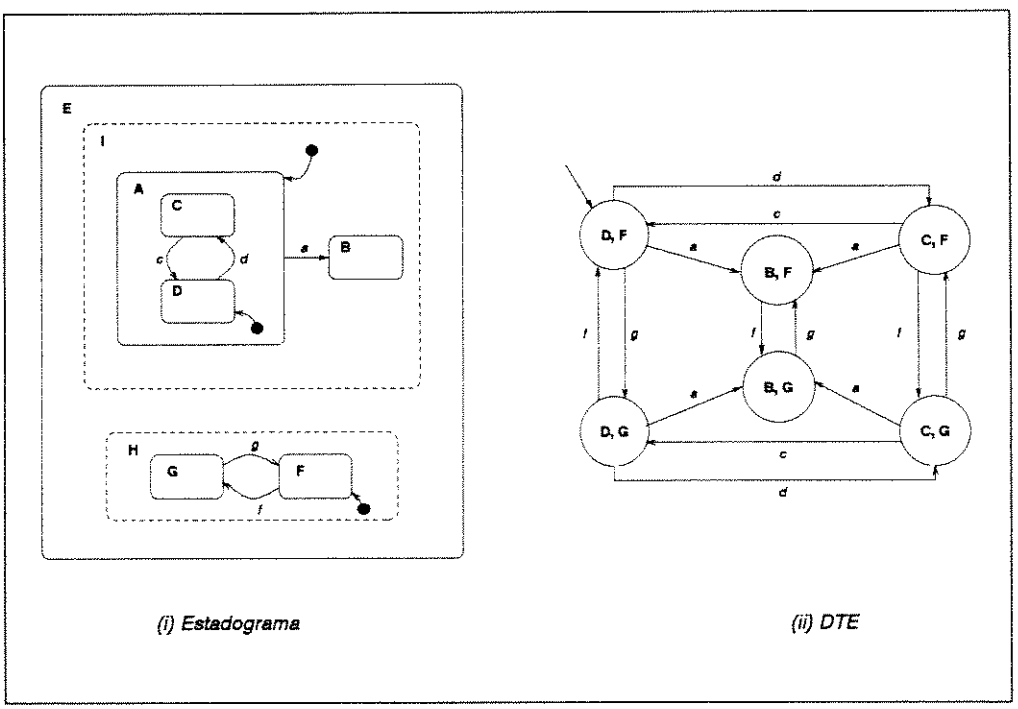

Figura 3.4: *Um exemplo comparativo de um estadograma e o DTE associado*

Com a utilização de estadogramas, pode-se identificar a necessidade do estado A, e especificá-lo posteriormente, ou identificar os subestados C e D e encapsulá-los no estado A. Para o estado B, não interessa o que acontece em A (modularidade). Observa-se também a hierarquia existente desde o estado E até o estado C e D. Nos diagramas convencionais

[6]Neste trabalho, o autor utiliza estadogramas, com adaptações significativas, na especificação e implementação de controle de diálogo de interfaces.

(*ii*), todas essas 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, poderia ser identificada a presença adicional de um *modo* (Seção 3.2.2). Utilizando-se estadogramas bastaria adicionar um estado concorrente ao conjunto de estados já especificados; ao contrário, com os diagramas convencionais é preciso duplicar toda a especificação. Isto poderia ser equivalente à inclusão do estado H no estadograma. É simples imaginar a especificação antes do acréscimo do estado H, o que já não é tão simples com os diagramas de estados convencionais (dificuldade de se expressar concorrência). Nos estadogramas, várias outras características ainda poderiam ser exploradas; por exemplo, comunicação entre estados concorrentes e transições condicionais.

3.4.5 Modelo de eventos

A maioria das técnicas para representação de diálogo assíncrono são variações de mecanismos baseados em eventos. O *modelo de eventos* é baseado no conceito de *eventos de entrada* encontrado em muitos pacotes gráficos. Nestes pacotes, os dispositivos de entrada são vistos como fontes de eventos. Cada dispositivo de entrada gera um ou mais eventos quando o usuário interage com ele, sendo colocados numa fila quando gerados. O programa de aplicação remove os eventos um de cada vez, através de chamadas a rotinas no pacote gráfico.

O *modelo de eventos* consiste numa extensão dessa idéia básica. Os eventos são gerados pelos dispositivos de entrada ou no componente de controle do diálogo (ver modelo Seeheim, Seção 2.10). O programador está livre para definir novos tipos de eventos mais apropriados para uma aplicação específica. No modelo de eventos não existe explicitamente uma fila de eventos; quando um evento é criado, ele é enviado para um *tratador de eventos*, que consiste de um processo (definido por um procedimento) capaz de manipular certos tipos de eventos. Este procedimento pode executar algum cálculo, gerar novos eventos, chamar rotinas da aplicação, criar novos tratadores de eventos, ou destruir tratadores existentes.

O comportamento de um tratador de eventos é definido por um *template*, que consiste na definição de parâmetros do tratador de eventos, suas variáveis locais, os eventos que ele pode processar, e os procedimentos associados a cada evento. No modelo de eventos uma interface com usuário é descrita por um conjunto de *templates*, como no exemplo da Figura 3.5.

Em [Green86] são apresentadas três técnicas para descrição de diálogo, baseadas geralmente em gramáticas, redes de transição e eventos. São apresentadas definições formais desses três modelos, com algoritmos para converter as notações em formas executáveis. Green conclui que os mecanismos baseados em eventos possuem o maior poder de descrição, pois podem ser usados para representar tanto diálogos seqüenciais como assíncronos; por exemplo, na edição de múltiplos arquivos, a tarefa de *cut-and-paste* entre arquivos diferentes pode ser feita facilmente enviando eventos de uma janela para a outra, o que não é possível com os outros dois modelos. São apresentados também algoritmos para conversão dos modelos de gramáticas e diagramas de transição para o modelo de eventos.

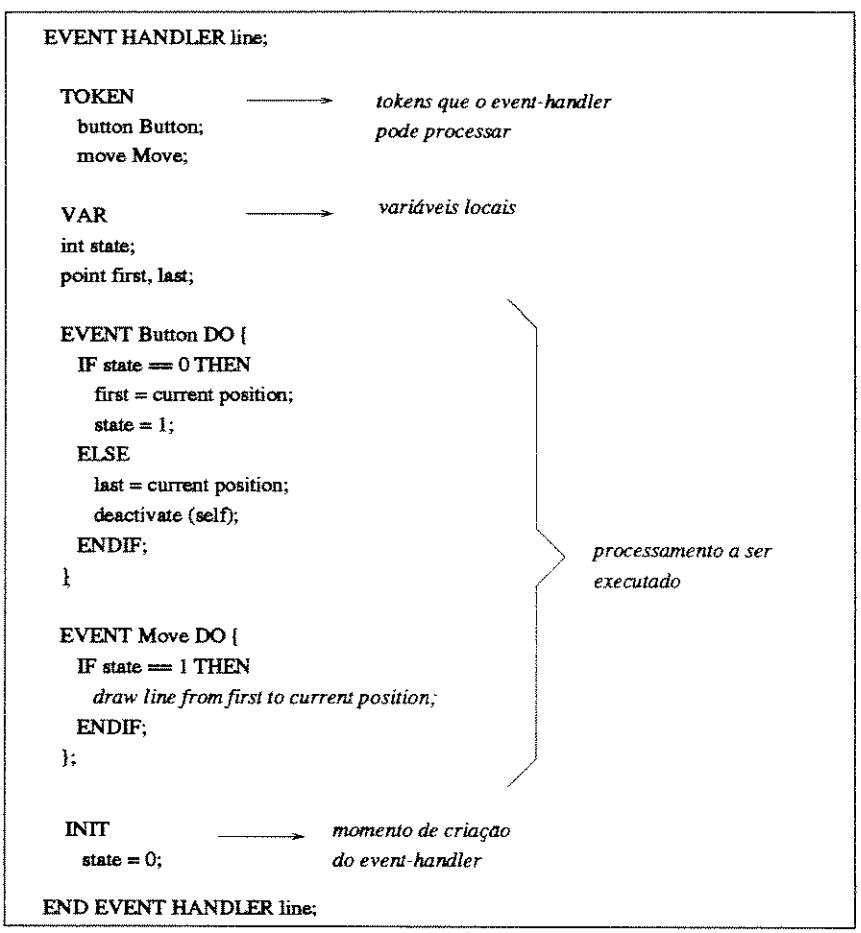

Figura 3.5: *Um tratador de eventos para o exemplo de rubber band line*

A principal desvantagem das linguagens baseadas em eventos está na dificuldade de se criar código correto, devido ao fluxo de controle não ser localizado(o usuário pode interromper várias ações de diálogo, além de dar continuidade a qualquer uma das ações interrompidas). Pequenas modificações em uma parte da especificação podem afetar muitas outras partes, e

o entendimento da especificação é dificultado à medida que esta cresce, pois não existe uma hierarquia definida.

3.4.6 Especificação gráfica

Alguns sistemas de desenvolvimento permitem que a definição da interface seja, ao menos parcialmente, feita através da colocação de objetos diretamente na tela [Myers89]. A filosofia por trás deste enfoque é que a apresentação visual da interface é o aspecto mais importante da especificação. O enfoque central das ferramentas para especificação gráfica é projetar a interface por *demonstração*, ao invés de especificação convencional. Esta técnica é muito mais vantajosa para o projetista, pois a interface é projetada visualmente, os comandos e ações da interface são dados graficamente.

A principal vantagem da utilização de sistemas que utilizam especificação gráfica está na facilidade de se gerar e testar protótipos, implicando na redução de tempo durante as fases de definição e especificação da interface. A principal desvantagem é que o resultado da especificação é a própria interface (ou parte dela); a especificação deixa de ser um veículo de comunicação entre as pessoas da equipe de desenvolvimento.

3.4.7 Especificação utilizando base de conhecimento

O entendimento de um sistema pelo usuário (*modelo do usuário*) é formado e continuamente refinado através do contato com a *imagem do sistema* (comportamento da interface). Pode haver erros nesse processo de comunicação: a concepção dos projetistas (*modelo de projeto*) pode ser bem diferente em relação à visão dos aspectos do mundo real pela comunidade usuária. Mark [Mark86], propõe um novo modelo conceitual, o *modelo da imagem do sistema*, que consiste num conjunto de conceitos abstratos de como os objetos e ações podem se comportar na interface. A *especificação baseada em conhecimento* oferece uma estrutura na qual um grupo de projetistas pode expressar suas idéias em termos de um *modelo da imagem do sistema* em comum.

A especificação utilizando base de conhecimento possui um enfoque bem diferente dos outros métodos de especificação conhecidos. Utiliza uma *base de conhecimento* consistindo de objetos, atributos, ações, pré e pós-condições nas ações (ver [Foley89]), que formam uma descrição declarativa de uma interface (estabelecem *o que* deve acontecer, e não *como* acontecer). A partir dessas descrições, interfaces alternativas são geradas para a mesma funcionalidade básica.

As principais vantagens dessa abordagem consistem na facilidade de manutenção da especificação e incorporação de novas características à interface, pois o sistema confere cada nova idéia na base de conhecimento, checando se ela está consistente com o que ele já conhece. Muita pesquisa ainda deve ser feita para tornar utilizáveis as técnicas de especificação baseadas em conhecimento. Os principais problemas residem na dificuldade de obtenção de mecanismos de inferência confiáveis, e o tempo de especificação geralmente ser maior que o dos métodos de especificação diretos.

3.5 Técnicas de decomposição de tarefas

Nas técnicas de representação discutidas até agora o que predomina geralmente é a visão da interface pelo sistema; os nodos nos *DTEs*, os eventos, as árvores de menus, geralmente estão associados a funções do sistema. Para se descrever a interface com base nas ações do usuário é necessário mudar o enfoque para abordagens orientadas a tarefas, que é a maneira como o usuário enxerga a interface. As técnicas apresentadas na seção anterior encaixam-se, de uma maneira geral, nos *modelos de síntese* de especificação, ou seja, não fornecem subsídios explícitos para avaliação de usabilidade da interface.

A seguir, são descritas algumas outras técnicas para representação da interface cujo enfoque está nas tarefas executadas pelo usuário. Algumas dessas técnicas são consideradas originalmente como *modelos de análise*, ou seja, o objetivo principal não é capturar a interface que está sendo projetada, mas construir uma representação da mesma, com o propósito de previsão de desempenho de usuários para a avaliação de usabilidade.

3.5.1 GOMS

O modelo *GOMS* (*Goals, Operators, Methods and Selection rules*), proposto em [Card83], é considerado o mais estudado dos modelos analíticos (é o que tem recebido mais testes empíricos). Alguns dos trabalhos mais recentes sobre aplicação do modelo *GOMS* podem ser encontrados em [Irving94, Gong94, Kieras94].

O princípio fundamental do modelo *GOMS*, como método de análise de tarefas, está baseado na previsão do comportamento do usuário. Para isto, é necessário analisar as tarefas por ele executadas, determinando os objetivos, operadores e restrições da tarefa. O comportamento de um usuário é descrito por uma seqüência de operadores de processamento

de informação, e o tempo que um usuário leva para executar uma tarefa é estimado pela manipulação dos tempos desses operadores individuais.

A seguir, são descritos os componentes do modelo *GOMS*:

- **Objetivos**: define um estado de desejos a serem realizados. A sua função dinâmica é prover um ponto de memória para o qual o sistema pode retornar falha, erros ou sucesso na obtenção do objetivo traçado;
- **Operadores**: são as ações elementares. Objetivos e Operadores geralmente possuem a forma *ação-objeto*; entretanto, um objetivo consiste em algo a ser realizado, enquanto um operador é simplesmente executado. Os operadores podem ser *externos* (o usuário troca informação com o sistema ou outros objetos no ambiente; por exemplo, o usuário lê uma página de um manual), *perceptuais* (procurar pelo cursor na tela), *motores* (pressionar uma tecla, movimentar o mouse) e *mentais* (não observáveis, inferidos pelo analista; por exemplo: tomar uma decisão básica, armazenar ou recuperar um item da memória). Os operadores definem a *granularidade da análise* (modelos podem ser construídos com diferentes níveis de detalhe);
- **Métodos**: descrevem um procedimento para realizar um objetivo. Um método consiste de um conjunto de operadores;
- **Regras de Seleção** (Estrutura de controle): Para realizar um objetivo, podem existir mais de um método disponível. As regras de seleção consistem de um controle para escolher o método apropriado. Essas regras são baseadas nos *sistemas de produção* e possuem a seguinte forma:

"se <*tais condições na tarefa atual são satisfeitas*>
então <*use o método M*>".

De forma mais sucinta, o modelo consiste numa descrição de *métodos* necessários para realização de *objetivos* especificados. Os métodos consistem de uma série de passos, ou *operadores* que o usuário executa. Se existir mais de um método para realizar um objetivo, o modelo inclui as *regras de seleção* para a escolha do método apropriado dependendo do contexto.

Kieras [Kieras88] comenta que a notação do modelo *GOMS* apresentada em [Card83] é difícil de usar. Ele propõe uma linguagem chamada *NGOMSL (Natural GOMS Language)* para expressar os modelos. Essa linguagem é relativamente fácil de ler e escrever, sem se ter

conhecimento de modelos de regras de produção. Também é proposto um procedimento para construção do modelo *GOMS*, usando métodos de expansão *top-down* (do objetivo mais geral para o mais específico) e *breadth-first* (considera todos os métodos que estão num mesmo nível de hierarquia) que parecem ser mais intuitivos e fáceis para descrição de estruturas de objetivos. Uma vez que o modelo *GOMS* tenha sido desenvolvido, previsões de aprendizado e desempenho de usuários podem ser obtidas, baseadas na abordagem *Keystroke-Level model* [Card83]. Recentemente, em [Haunold94], o modelo *Keystroke* foi utilizado na avaliação de uma aplicação gráfica.

Após calcular estimativas de desempenho, o analista deve rever o projeto. Algumas sugestões apresentadas em [Kieras88] são:

- garantir que os objetivos mais freqüentes sejam fáceis de aprender;
- procurar eliminar, reescrever e combinar métodos;
- eliminar a necessidade de recuperar operadores da memória de longo prazo (*LTM*);
- não reduzir o número de *keystrokes*, se isto implica num aumento do número de operadores mentais.

Recentemente, tem havido tentativas de incorporar conceitos de *lógica nebulosa* e *teoria de conjuntos nebulosos* em pesquisa *HCI*. Em [Card83] os autores sugeriam a incorporação de elementos estocásticos ao modelo *GOMS* (tempos de operadores expressos como distribuição de probabilidades e regras de seleção probabilísticas). Karwowski e Salvendy [Karwowski92] vão um pouco mais longe e sugerem que o modelo *GOMS* poderia tratar de imprecisão[7] nas regras de seleção. Dessa forma, o modelo *GOMS* é estendido , permitindo componentes que podem assumir valores precisos, probabilísticos ou nebulosos (os objetivos, operadores e métodos podem ter valores precisos ou nebulosos, enquanto as regras de seleção podem ser expressas de maneira probabilística ou nebulosa). Os autores fizeram um experimento para validar o *modelo GOMS nebuloso* e obtiveram resultados bastante satisfatórios. Entretanto, muita pesquisa ainda é necessária para explorar de uma maneira mais completa a potencialidade da teoria nebulosa na área de fatores humanos.

[7] Enquanto a teoria de probabilidades lida com *incerteza*, a teoria de conjuntos nebulosos lida com *imprecisão*.

A abordagem *GOMS* é mais aplicável para tarefas que possam ser descritas em termos de uma estrutura hierárquica de objetivos e sub-objetivos; por exemplo, tarefas que descrevam uma planilha eletrônica ou um editor de textos. O modelo *GOMS* também se encaixa melhor para tarefas seqüenciais, onde o usuário executa uma subtarefa de cada vez e comete poucos erros. Portanto, tarefas que envolvam processamento paralelo, não tenham hierarquia bem definida e sujeitas a erros freqüentes de usuários não são boas candidatas para modelagem *GOMS*.

3.5.2 TAG

Task-Action Grammar (*TAG*) é um formalismo para representação e avaliação de linguagens de descrição de tarefas (especificação de ações) [Payne89]. É um modelo que enfatiza as interfaces baseadas em linguagens de comandos.

Um dos principais objetivos para o projetista é a obtenção de consistência na interface. *TAG* ataca o problema de consistência e tenta explicar porque especificações consistentes implicam em interfaces mais fáceis de aprender e usar. Para capturar essas noções de consistência, Reisner [Reisner81] (ver Seção 3.4.1) propôs uma *gramática de ações* para descrever duas versões da interface de um sistema gráfico. Ela mostrou que a versão que tinha uma gramática mais simples era mais fácil de aprender. Payne e Green [Payne86] expandiram o trabalho de Reisner, dando um maior enfoque na noção de consistência, através da notação *TAG*.

A Figura 3.6 apresenta um fragmento de uma especificação em *TAG*. O ponto principal que se deve notar é a procura de generalização de um conjunto de comandos da interface, facilitando a checagem de consistência[8].

[8] Existem várias formas de consistência e, algumas vezes, *inconsistência* pode ser uma característica desejável na interface (por exemplo, para chamar a atenção do usuário em operações que possam causar danos inesperados).

```
TASK FEATURES
Old  file, disk, null
New    file, disk, null

SIMPLE TASKS
Copy file {Old = file, New = file}
Copy disk {Old = disk, New = disk}
Delete file {Old = file, New = null}
Create file {Old = null, New = file}

TASK-ACTION RULE SCHEMAS
Task [Old, New] --> name [Old, New] +
                              par1 [Old] + par2 [New]
name [Old = file, New = file] --> "copy"
name [Old = file, New = null] --> "erase"
name [Old = null, New = file] --> "edit"
name [Old = disk, New = disk] --> "diskcopy"
par1 [Old = null] --> NULL
par1 [Old = file] --> drive-id + "filename"
par1 [Old = disk] --> drive-id
par2 [New = null] --> NULL
par2 [New = file] --> drive-id + "filename"
par2 [New = disk] --> drive-id
drive-id --> NULL | "A:" | "B:" | "C:" | "D:"
```

Figura 3.6: *Descrição em TAG de algumas operações da linguagem de comandos do IBM-PCDOS*

As descrições em *TAG* não enfatizam a representação do fluxo de controle durante a execução; elas simplesmente representam a decomposição de um problema em tarefas simples, e como expressar essas tarefas em ações.

3.5.3 CLG

O formalismo *CLG* (*Command Language Grammar*), proposto em [Moran81], é um modelo orientado a tarefas e também um modelo para representação de diálogo. O modelo cria uma estrutura para descrever muitos aspectos da interface. Moran descreve como *CLG* pode ser considerado de três pontos de vista diferentes:

1. **Visão lingüística**: enxerga *CLG* como uma análise da estrutura da interface do sistema (*Modelo estrutural*).

2. **Visão psicológica**: enxerga *CLG* como uma maneira de descrever o conhecimento que o usuário final tem sobre o sistema (*Modelo de análise de tarefas*).
3. **Visão de projeto**: *CLG* é vista como um mecanismo de representação para o projeto do sistema (*Modelo de representação*).

CLG particiona a interface em três maiores componentes. Cada componente é dividido em níveis, cada um como refinamento do anterior:

- **Componente conceitual**:
  nível de tarefa
  nível semântico
- **Componente de comunicação**:
  nível sintático
  nível de interação
- **Componente físico**:
  nível de layout
  nível de dispositivos

A descrição de cada nível contém procedimentos, escritos em uma notação parecida com linguagem de programação em alto nível, que descrevem as tarefas realizadas pelo sistema, em termos de ações disponíveis em determinado nível, através de um processo de refinamentos sucessivos. Em [Firth91] é apresentado detalhadamente um exemplo de aplicação do formalismo *CLG* para descrição da interface do utilitário *NotePad* do *MacIntosh*. Esta especificação revelou falhas no projeto, que não haviam sido previstas, e resultou numa reestruturação no nível de tarefa.

A maior vantagem do formalismo *CLG* é que ele consiste de uma *abordagem em camadas*, começando com o modelo conceitual da interface. Como desvantagem, *CLG* foi originalmente proposto para descrever linguagens de comandos; portanto, é considerado pobre para descrição de aspectos dinâmicos de interfaces *WIMP*.

3.6 Escolha de uma técnica de especificação

No projeto de software convencional não existe uma técnica de especificação para todo e qualquer tipo de aplicação. Da mesma forma, a especificação de projeto de interfaces, atualmente, não compreende o uso de apenas uma linguagem de especificação, mas sim, de várias linguagens focalizadas sobre aspectos distintos.

O ideal seria aplicar uma linguagem de especificação que fornecesse diretamente resultados da avaliação semântica da interface. Entretanto, muita pesquisa ainda é necessária para que as técnicas baseadas em modelos de análise sejam universalmente aceitas pela comunidade de projetistas de *HCI*. Muitos pesquisadores ainda questionam a efetividade desses modelos para o projeto de interfaces. Em [Gugerty93] são apresentados alguns problemas com os modelos analíticos atuais:

- aplicam-se somente a tarefas e contextos específicos, para os quais eles foram desenvolvidos, sendo difícil a aplicação para novas interfaces;
- podem não se adequar à maneira intuitiva e informal de os projetistas de interface geralmente trabalharem;
- geralmente consomem muito tempo e dinheiro.

As técnicas do domínio comportamental descrevem a interação do ponto de vista do usuário e geralmente são orientadas a tarefas. *GOMS*, *TAG* e *CLG* permitem ao projetista descrever tarefas em vários níveis de abstração. O modelo *GOMS* é muito importante para a análise de tarefas; entretanto, o tamanho da descrição gerada pode ser enorme. *CLG* oferece uma estrutura para descrever muitos aspectos da interface, entretanto, a descrição em cada nível contém procedimentos escritos numa linguagem parecida com linguagem de programação de alto nível. *TAG* enfoca muitos aspectos de consistência na interface mas, da mesma forma que nas gramáticas, o fluxo de controle não é facilmente observável.

Os modelos de decomposição de tarefas *GOMS*, *CLG* e *TAG* enfocam principalmente a especificação de objetivos do usuário, e a tradução desses objetivos em ações, conforme os estágios de atividades do usuário envolvidos na execução de uma tarefa [Norman86b]. Esses modelos não representam associação direta dos objetivos e ações com o feedback e os estados da interface. Além disso, são modelos de execução de tarefa num tempo contínuo, não oferecem suporte à representação de interrupção, intercalação e relacionamento entre tarefas concorrentes.

As representações por gramáticas (*BNF*) tendem a ser do domínio comportamental, pois geralmente descrevem expressões do usuário da interface; entretanto, são difíceis de ler e escrever. Os *DTEs* possuem a vantagem de mostrar o fluxo de controle explicitamente numa forma gráfica. São considerados mais voltados para o domínio de construção, pois constituem uma representação direta das funções do sistema (por exemplo, o sistema está no estado X, se o usuário fornecer a entrada A, o sistema faz uma transição para o estado Y). Estadogramas e *DTEs* são muito bem empregados para representar informação de mudança de estado na interface, mas não representam satisfatoriamente o feedback ou aparência da tela em interfaces *WIMP*.

Os tratadores de eventos são utilizados para representar eventos que são resultantes das ações do usuário. Possuem poder de expressão maior que *BNF* ou *DTEs*, principalmente por suportarem especificação de tarefas concorrentes. Por representarem a interface com a visão do sistema (por exemplo, procedimentos a serem chamados em resposta a um evento) acabam quase sendo a implementação do software da interface, de difícil entendimento para a equipe que desenvolve a interação.

Baseando-se nessas observações e nas reais necessidades identificadas no momento da escolha de uma notação para descrever a interface que pretendia-se especificar, decidiu-se que o enfoque seria nos modelos de síntese de interface, onde uma descrição de tarefas do ponto de vista do usuário fosse o principal resultado da aplicação da metodologia. Mas isto não era o suficiente, pois procurava-se uma notação que fosse concisa, precisa, não ambígua e fácil de utilizar, pois era interessante obter uma especificação que não fosse somente para uso exclusivo do projetista da interface, mas que servisse de veículo de comunicação entre as diversas pessoas envolvidas no projeto da interface.

No próximo capítulo apresenta-se a técnica *UAN* (*User Action Notation)* de representação de projeto de interface com o usuário. Essa foi a notação básica utilizada neste trabalho. Alguns dos motivos da escolha da notação *UAN* foram os seguintes:

- suporta diretamente *síntese* de interfaces (criação e documentação). Entretanto, como é apresentado no próximo capítulo, *UAN* também pode ser utilizada para analisar a estrutura de tarefas e detectar possíveis ambigüidades e inconsistências no projeto;
- é uma técnica para representar o projeto da interação, e não o projeto do software da interação;
- fornece suporte à especificação de interfaces de manipulação direta;

- apresenta de forma simples suporte para especificação de relacionamentos temporais entre tarefas; por exemplo, concorrência, intercalação e interrupção.
- combina boas características de várias técnicas que foram apresentadas neste capítulo, por exemplo, *DTEs* e modelo de eventos;
- produz especificações concisas, precisas, não ambíguas e fáceis de entender, após pouco tempo de uso, facilitando a comunicação entre os membros da equipe de desenvolvimento da interface;
- utiliza a natureza gráfica dos *DTEs* para representação dos estados da interface;
- oferece suporte para especificação do feedback da interface para as ações do usuário.

3.7 Resumo

Neste capítulo foram apresentadas várias técnicas para representação do projeto de interfaces. Procurou-se enfatizar apenas as características mais importantes em cada uma delas; maiores detalhes e outros exemplos podem ser encontrados nas referências fornecidas. Apresentou-se uma comparação geral entre as principais técnicas e os motivos da escolha da técnica *UAN* (*User Action Notation*) como principal objeto de estudo neste trabalho. Esta técnica é descrita em maiores detalhes (sintaxe e semântica) no próximo capítulo.

Capítulo 4

UAN - Uma técnica para especificar projetos de interação

UAN é uma notação compacta, poderosa;
uma abordagem de alto nível para especificar o comportamento
de sistemas interativos e descrever ações do usuário.
[Shneiderman92], p. 517

O principal objetivo neste capítulo é fornecer uma base para o entendimento de descrições *UAN* de interfaces. Esta foi a notação básica utilizada para especificação da interface a ser discutida neste trabalho. É fornecido um breve histórico, origem, descrição da notação e um exemplo de sua aplicação. Também é examinado brevemente o relacionamento dessa notação com o *ciclo de vida estrela* (Seção 2.5) e com os seis focos para decisões de projeto apresentados no *modelo snowflake* (Seção 2.6.1).

4.1 Origem e utilização da UAN

A *UAN* é uma notação orientada a ações e tarefas do usuário, desenvolvida para descrever o projeto de interfaces de manipulação direta[1]. Essa notação foi proposta inicialmente para auxiliar projetistas e programadores do *Dialogue Management Project*[2]. O idealizador da *UAN*, Antonio C. Siochi, propôs a notação, baseado nos problemas constantes de volume de especificação que estava sendo gerado para descrever o projeto da interface, o que

[1] As abordagens de especificação baseadas em gramáticas e diagramas não expressam convenientemente a variedade de ações permissíveis e o *feedback* visual que as interfaces de manipulação direta fornecem.

[2] desenvolvido no Departamento de Ciência da Computação do *Virginia Polytechnic Institute and State University*.

dificultava a comunicação entre a equipe de projetistas e programadores.

Desde a sua criação, muitas pessoas têm utilizado a *UAN* na descrição de diversos tipos de interfaces, contribuindo assim para a sua extensão e formalização. Em particular, algumas das instituições que têm utilizado a *UAN* de forma experimental são:

- Laboratório de Interfaces do *Texas Instruments* (Dallas, Texas), em projetos de interfaces para telefonia;
- *Naval Surface Warfare Center* (Virginia), na documentação de vários *guidelines* de construção de interfaces;
- *NCR Corporation*, na representação de interfaces multimídia;
- *Universidade de Glasgow* (Escócia), no projeto de interfaces para várias aplicações;
- Laboratórios da *DEC*, no projeto da interface de um editor gráfico;
- Laboratório da *Jet Propulsion*;
- *Laboratoire de Génie Informatique, Institut IMAG* (Grenoble, França), na descrição de uma *interface multimodal* para um sistema de informação sobre viagens aéreas.

4.2 Visão geral da UAN

As descrições de tarefa utilizando *UAN* devem ser escritas primeiramente por alguém que esteja projetando o *componente de interação* de uma interface e, então, ser lida por todas as outras pessoas que desenvolvem a interface, particularmente aquelas que estejam projetando e implementando o software da interface com o usuário. Se os detalhes da interação não são passados para o programador, este fica livre para tomar decisões a respeito do que o projetista pretendia, ou terá que procurá-lo para perguntar como será a interação. Nenhuma destas situações é desejável, pois a interface do produto final pode apresentar inconsistências e a usabilidade final do sistema ficar comprometida.

Uma técnica como *UAN* consiste de um mecanismo apropriado para registrar as decisões do projeto de interação *precisamente, concisamente, sem ambigüidades, e em detalhe* [Hix93]; assim as decisões a respeito do projeto de interação não são deixadas para o programador. Outra justificativa para a utilização de *UAN* é que ela força a equipe a documentar as decisões do projeto da interface, facilitando uma posterior alocação de novas pessoas à equipe de desenvolvimento da interface. Normalmente as pessoas gastam boa parte do tempo

para se *envolverem* com o projeto e uma abordagem estruturada como *UAN* pode ajudar na comunicação entre os membros da equipe de desenvolvimento, reduzindo mal-entendidos e confusões sobre o projeto.

A *UAN* é uma notação *orientada a tarefas*, que descreve o comportamento do usuário e da interface durante uma interação. Uma interface é representada como uma estrutura hierárquica de tarefas *assíncronas*. As ações físicas do usuário necessárias para realizar uma tarefa, o feedback correspondente da interface e as informações de mudança de estado da interface são representadas num nível de abstração mais baixo, que denominou-se *nível articulatório* [3]. Níveis de abstração mais altos, ou *níveis semânticos*, escondem esses detalhes e são necessários para construir a estrutura de tarefas na interface. Em todos os níveis, ações do usuário e tarefas são combinadas com *relações temporais* (Seção 4.5) tais como seqüenciamento, intercalação e concorrência.

As descrições em *UAN* são apresentadas em forma de tabela, com o *layout* básico apresentado na Figura 4.1. A leitura da *tabela de especificação UAN* deve ser feita de cima para baixo e da esquerda para a direita. Na Seção 4.4 discutem-se as informações que são representadas em cada uma das colunas dessa tabela de especificação.

Tarefa: <*nome-da-tarefa*>			
Ações do Usuário	**Feedback da Interface**	**Estados da Interface**	**Conexão com a Aplicação**
⋮	⋮	⋮	⋮

Figura 4.1: *Layout básico da tabela de especificação UAN*

4.3 Um exemplo de descrição em UAN

Antes de descrever a sintaxe da notação *UAN*, apresenta-se um exemplo de sua utilização na descrição de uma interação num aplicativo hipotético denominado *FManager*[4]. Este exemplo serve de referência para apresentação da *UAN* básica discutida neste capítulo.

[3] relacionado com a *distância articulatória* discutida em [Norman86b].
[4] similar ao utilitário *FileManager*, do *OpenWindows 3.0*

Descrição prosaica da interação:

Nível de Tarefas:

1. Expandir o ícone da aplicação.
2. Movimentar a janela da aplicação.
3. Apagar um arquivo específico.
4. Renomear um arquivo específico.

Nível de Ações do Usuário:

1. Expandir o ícone da aplicação.
   (a) Mover o cursor para o ícone de uma aplicação específica.
   (b) Dar um clique duplo no botão esquerdo do mouse, sobre o ícone da aplicação. Ao liberar o botão, o ícone é selecionado, indicado por um destaque (contorno), e a janela da aplicação é ativada e imediatamente mostrada na tela. O ícone deve desaparecer da tela, sendo *expandido* para a janela da aplicação.
2. Movimentar a janela da aplicação.
   (a) Mover o cursor para uma das bordas da janela da aplicação.
   (b) Pressionar o botão esquerdo do mouse sobre uma das bordas da janela, que deverá ser destacada com um contorno nas bordas.
   (c) Com o botão esquerdo do mouse pressionado, mover o cursor pela tela. Um contorno da janela deve acompanhar o movimento do cursor.
   (d) Liberar o botão do mouse (a janela se move para a nova posição).
3. Apagar um arquivo específico.
   (a) Mover o cursor para o ícone do arquivo a ser apagado.
   (b) Pressionar o botão esquerdo do mouse sobre o ícone do arquivo, que deverá ser destacado com uma mudança de cor.
   (c) Mantendo o botão do mouse pressionado, movimentar o cursor para arrastar a representação do ícone do arquivo para o contexto do *trashicon* (*ícone cesta de lixo*).

(d) Liberar o botão do mouse; o ícone do arquivo desaparece. Algum tipo de destaque no ícone cesta de lixo deve ser dado para indicar que o arquivo ainda pode ser recuperado.

4. Renomear um arquivo específico.

   (a) Mover o cursor para o nome do arquivo a ser renomeado.

   (b) Dar um clique simples no botão esquerdo do mouse, sobre o nome do arquivo. Um tipo de destaque deve ser dado no ícone que representa o arquivo selecionado, e o nome do arquivo deve ser colocado numa linha de edição.

   (c) Editar o novo nome do arquivo, finalizando com a tecla *return*. Ao iniciar a edição do novo nome, o nome antigo deve ser apagado.

Descrição da interação utilizando *UAN*:

Nível de Tarefas:

Expandir o ícone da aplicação
(Movimentar a janela da aplicação
Apagar um arquivo específico |
Renomear um arquivo específico)*

Nível de Ações do Usuário: (Figura 4.2)

Ações do Usuário	Feedback da Interface	Estados	Conexões
Tarefa 1: *Expandir o ícone de uma aplicação*			
~[applicon'];			
$M_L \vee \wedge \vee \wedge$	applicon'-!:applicon'! display(applwindow') erase(applicon') (see figure x)	selected=appl	display the applwindow at the last position it appeared
Tarefa 2: *Movimentar a janela de uma aplicação*			
~[applwindow']$_{\text{border}}$; $M_L \vee$	applwindow'-!:applwindow'!	selected = applwindow	
~[x,y]* ~[x',y']	outline(applwindow') >~		
$M_L \wedge$	@x',y' redisplay(applwindow')	selected = null	display the applwindow at the new position
Tarefa 3: *Apagar um arquivo*			
~[fileicon']; $M_L \vee$	fileicon'-!:fileicon'! (see figure y)	selected = file	
~[trash icon];	outline(fileicon') >~ trashicon!		
$M_L \wedge$	erase(fileicon') erase(outline(fileicon')) trashicon! (see figure z)	selected = null	mark file to be deleted
Tarefa 4: *Renomear um arquivo*			
~[filename']; $M_L \vee$	filename'-!:filename'!	selected = filename	
K(newfilename = (a-z)(a-z|0-9)$^+$)	erase(filename')		call the line editor
return$\vee \wedge$	display(newfilename)	selected = null	rename the file

Figura 4.2: *Um exemplo de descrição de interação utilizando UAN*

4.4 Descrição da notação básica

Nesta seção são apresentados os principais conceitos e características presentes na notação *UAN* e a simbologia básica utilizada por essa notação. A descrição que se segue está baseada em [Siochi89, Hartson90, Siochi91, Hartson92a, Hix93]. Essas referências também apresentam vários exemplos de especificação de interação utilizando *UAN*.

4.4.1 Ações do Usuário

Nesta seção são apresentadas algumas características da notação que enfatizam as descrições na coluna de *Ações do Usuário* na tabela de especificação *UAN* (Seção 4.1).

Dispositivos e ações primitivas do usuário

A *UAN* possui símbolos específicos[5] para representar dispositivos e ações comuns do usuário. Os dispositivos são os meios através dos quais o usuário interage com o sistema. Geralmente encontram-se numa das seguintes classes:

- dispositivos tipo *interruptor*, que podem ser pressionados e liberados, causando transmissão de um simples caráter ou sinal. Nesta classe podem ser citados, por exemplo, o mouse (M_L, M_C, M_R) respectivamente representam os botões esquerdo, central e direito de um *mouse de três botões*), a tecla *Esc* e as *teclas de função*;
- dispositivos através dos quais as ações do usuário resultam em cadeias de caracteres. Exemplos são o teclado (K, de *keyboard*) e dispositivos de reconhecimento de voz;
- dispositivos não usuais como pedais, joysticks e *eye trackers*.

Os símbolos para especificar as ações do usuário são escolhidos de forma a representar uma *mímica*[6] das ações, tais como $\vee$ e $\wedge$ para as ações de pressionar e liberar um interruptor, respectivamente, $\sim$ para a ação de movimento do cursor na tela, $>\sim$ para a ação de *arrastar*[7] um objeto pela tela. É importante ressaltar que a notação *UAN* está em aberto, podendo-se criar os símbolos próprios de acordo com as necessidades, padrões e estilos de cada equipe de desenvolvimento. Cabe ainda lembrar que não é objetivo neste trabalho criticar a notação em relação a detalhes psicológicos ou semióticos dos símbolos que ela apresenta; por exemplo,

[5] Na Seção 4.6 apresenta-se uma relação completa dos símbolos utilizados.
[6] Os símbolos apresentados são *mnemônicos sugestivos* das ações, chamados de *onomatopéias visuais*.
[7] pressionar o botão do mouse e mantê-lo assim enquanto o cursor é movido

por que não é utilizado um simples /, ou MOVE TO, ao invés de um ~ para representar a movimentação do cursor.

As ações do usuário feitas diretamente nos dispositivos de hardware (ex.: mover o cursor, pressionar o botão direito do mouse) são consideradas *ações primitivas*, pois geralmente não precisam ser decompostas em maiores detalhes. Para identificar cada uma das ações primitivas, o projetista deve primeiramente identificar cada dispositivo, e as ações físicas do usuário que se aplicam a esses dispositivos.

No exemplo da Seção 4.3 são apresentadas algumas ações do usuário, com as respectivas representações em *UAN*. Por exemplo, ~`[applicon']` significa mover o cursor para o ícone de uma aplicação, M_LV descreve a ação de pressionar o botão esquerdo do mouse, e `K(newfilename)` representa a entrada de dados via teclado. Nesse último caso, também é descrita uma sintaxe para entrada dos dados, ou seja, o nome do arquivo deve começar com uma letra, seguido ou não de qualquer combinação alfanumérica (ver descrição da tarefa 4 na Figura 4.2).

É importante observar que a ação de mover o cursor para um objeto X qualquer, representada por ~X, é feita de maneira independente do dispositivo que está sendo utilizado. A notação não indica como isso é feito, por exemplo, através de um mouse, *joystick*, *trackball* ou setas do teclado. Essa é uma importante característica da notação, pois caso seja mudado o dispositivo, as especificações não precisam ser alteradas globalmente.

Objetos e Contextos

No nível de descrição da aparência da interface, um objeto pode ser, por exemplo, um ícone, um botão do mouse ou uma janela. Objetos também podem ser específicos de cada aplicação, por exemplo, uma esfera (numa aplicação de modelagem de sólidos), uma aeronave (num jogo espacial), ou um parágrafo (num editor de textos). Devido à generalidade do nome de um objeto (por exemplo, o objeto *ícone de um arquivo*), um apóstrofo ' é adicionado para referenciar um objeto específico, distinguindo-o dos demais.

O *contexto de um objeto* é aquele através do qual ele é manipulado. Em *UAN*, o contexto é representado por símbolos de colchetes [ ] envolvendo o nome do objeto. Normalmente, o contexto de um objeto é ele próprio, por exemplo, para mover um ícone pela tela, basta posicionar o cursor em qualquer região do ícone e *arrastá-lo* para a posição desejada. Às vezes, o significado do contexto de um objeto pode ser dependente da tarefa que está sendo realizada, podendo acontecer do contexto de um objeto ser apenas parte dele, por exemplo,

para aumentar o tamanho de uma linha numa aplicação gráfica, o usuário deve posicionar o cursor numa das extremidades da mesma para efetuar a operação.

Retomando o exemplo da Seção 4.3, são apresentados alguns tipos de contexto de objetos, por exemplo, o contexto de um ícone de arquivo `[fileicon']` e contexto de um nome de arquivo `[filename']`. No caso da tarefa 2, de movimentação de janela, a parte manipulável do objeto é uma de suas bordas, representando-se em *UAN* por $\texttt{[applwindow']}_{\texttt{border}}$.

Repetição de tarefas

Numa determinada interação, o usuário pode executar uma mesma tarefa repetidas vezes. Em *UAN* utilizam-se os símbolos * (zero ou mais repetições), + (pelo menos uma repetição) ou o número específico de repetições.[8]

Na tarefa 2 (*movimentação da janela da aplicação*) do exemplo da Seção 4.3, foi necessário representar a repetição de uma ação do usuário. $\sim$`[x,y]` indica movimento do cursor para uma posição arbitrária da tela, enquanto a expressão $\sim\texttt{[x,y]}^*$ refere-se à movimentação do cursor zero ou mais vezes. Uma variante dessa expressão é representada por $\sim\texttt{[x,y]}^+$, que significa mover o cursor para uma ou mais posições da tela, isto é, pelo menos uma vez.

Mudança de intenção

Do ponto de vista de desenvolvimento de uma interface homem-computador, uma tarefa do usuário consiste numa seqüência de ações e subtarefas, sendo que a última ação marca o final da tarefa (*fechamento da tarefa*). Quando, por algum motivo, o usuário muda a sua intenção, não executando todas as sub-tarefas até atingir o fechamento da tarefa, diz-se que houve uma *interrupção*. Esses pontos onde uma tarefa pode ser interrompida é indicado nas descrições *UAN* através de um sinal de *ponto e vírgula*, como mostrado na coluna de ações do usuário do exemplo da Seção 4.3. Por exemplo, na tarefa 3 (*Apagar arquivo*), o usuário pode simplesmente movimentar o cursor para o ícone do arquivo, e não pressionar o botão esquerdo do mouse, sendo representado em *UAN* por $\sim$`[fileicon'];` $M_L\vee$. A especificação desses pontos de interrupção no momento em que se está projetando a interface podem ajudar bastante os programadores da interface, pois indicam pontos nos quais eles deverão manipular eventos especiais do usuário.

[8] Alguns símbolos da *UAN* foram trazidos das *linguagens de expressões regulares*.

Condições de viabilidade

Às vezes, um conjunto de ações ou tarefas só pode ser executado quando um conjunto particular de condições for satisfeito. Nesse caso, uma *condição de viabilidade* pode ser especificada. A forma geral deste tipo de expressão em *UAN* é uma condição com o sinal de *dois pontos* separando-a das ações (ou tarefas) a que ela se aplica, da seguinte forma:

condição : (ação1 ação2 ... ação n)

Uma condição de viabilidade atua como uma pré-condição que deve ser satisfeita para que as ações do usuário dentro do escopo sejam executadas como parte da tarefa que está sendo descrita. As condições de viabilidade também podem ser utilizadas na especificação do feedback da interface (Seção 4.4.2).

Uma maneira de simplificar a representação de condições de viabilidade em *UAN* é colocar a condição de forma implícita na realização da ação, por exemplo ∼`[applicon']` (no exemplo da Seção 4.3) significa mover o cursor para um ícone de uma aplicação específica, supondo que tal ícone já esteja no *workspace*[9].

É importante ressaltar que se a pré-condição não é atendida não implica que as ações não possam ser realizadas; elas até podem, porém o resultado pode não ser o esperado. Por exemplo, num editor de textos, se uma palavra estiver marcada e o usuário pressionar a tecla *Del*, a palavra é apagada. Entretanto, se a tecla *Del* for pressionada sem que haja uma palavra marcada, essa ação pode não causar qualquer efeito na interface.

4.4.2 Feedback da interface

O feedback da interface faz parte do projeto de interação e sua inclusão nas representações permite uma descrição mais completa do comportamento do usuário e da interface, enquanto interagem. O feedback é descrito na segunda coluna da tabela de especificação (Figura 4.1), associando-se cada ação do usuário com a resposta visual (ou sonora) fornecida pela interface. Apesar do feedback ser mostrado pelo sistema (*domínio de construção*), ele é incluído nas *descrições comportamentais*, pois está diretamente ligado às ações do usuário.

O feedback da interface é uma maneira de chamar a atenção do usuário, enfatizando uma mudança de estado em resposta a uma ação do usuário ou do sistema. Em *UAN*, o

[9] ambiente global de trabalho onde as aplicações são apresentadas.

símbolo de exclamação (!) é utilizado para representar, de uma maneira geral, o destaque de um objeto na coluna de *feedback da interface*. O símbolo -! após o nome de um objeto significa retirar o destaque, ou indica que o objeto não se encontra destacado.

No feedback da interface podem ser utilizados quantificadores ($\forall$), e operadores relacionais ($=$, $\neq$) para referenciar os objetos e ações sobre os mesmos. Esta simbologia ajuda a descrever o *comportamento mutuamente exclusivo* de algumas operações. Por exemplo, quando seleciona-se um ícone de arquivo no *FManager*, *clicando* o botão esquerdo do mouse sobre o mesmo, deve-se retirar o destaque de todos os outros ícones de arquivo que estiverem selecionados. A expressão ($\forall$ `icon` $\neq$ `icon':icon-!`) significa que, para todos os ícones, exceto aquele que foi selecionado, retirar o destaque.

Conforme comentado na Seção 4.4.1, as condições de viabilidade também podem ser utilizadas na especificação do feedback da interface. A expressão `fileicon'-!:fileicon'!`, na tarefa 3 do exemplo apresentado, indica que antes de destacar o ícone, deve-se testar se ele já não está destacado.

Alguns tipos de feedback

Existem vários tipos de feedback dependendo do tipo de objeto utilizado ou operação que estiver sendo executada, por exemplo: gerar um contorno em volta do objeto, na seleção de ícones e janelas; emitir um sinal sonoro, para chamar atenção em alguma operação perigosa; mudar de cor, usar um símbolo de *check* ($\surd$), numa caixa de diálogo; usar vídeo reverso, no destaque de um parágrafo marcado.

Para representar um objeto se movendo pela tela enquanto é arrastado pelo usuário, a *UAN* possui o símbolo $>\sim$, que sugere algo que acompanha o movimento do cursor. No exemplo da Seção 4.3 é mostrado esse tipo de feedback nas descrições das tarefas 2 e 3. O *rubberbanding*[10] é representado em *UAN* pelo símbolo $>>\sim$.

O projetista da interface pode criar símbolos sugestivos para representar o feedback na interface, ou definir nomes para representá-lo. Esses nomes funcionam como funções aplicadas a determinados objetos, por exemplo: `display(object')` para apresentar um objeto na tela, `erase(object')` para apagar um objeto da tela, `redisplay(object')` para apagar o objeto de uma posição e mostrá-lo em outra (como na tarefa 2 do exemplo apresentado).

[10] situação onde um objeto muda de tamanho à medida em que é arrastado pela tela.

4.4.3 Estados do sistema

Além de representar as ações do usuário e o feedback da interface, as descrições de tarefas em *UAN* devem especificar as mudanças de *estado do sistema* (terceira coluna da tabela de especificação *UAN* , na Figura 4.1), pois a interação pode ser diferente conforme os diferentes estados em que se encontra o sistema. No exemplo da Seção 4.3, pode-se observar que os estados estão relacionados com o tipo de objeto selecionado em cada tarefa descrita.

4.4.4 Conexão com a aplicação

Na coluna de *conexão com a aplicação* o projetista pode especificar as conexões das tarefas da interface com o componente computacional associado. Os projetistas e programadores no domínio de construção implementam estas observações, por exemplo, como rotinas *callback* a partir dos *widgets* da interface.

Retomando o exemplo da Seção 4.3, pode-se observar algumas indicações para a equipe de programadores da interface na última coluna da tabela. Por exemplo, na tarefa 3 é indicado que o arquivo deve ser previamente marcado, para posterior eliminação. Essa observação sugere um campo *booleano* na estrutura de registro do arquivo, indicando se o mesmo está ou não marcado para ser eliminado.

4.5 Utilização da UAN em níveis mais altos de abstração

A *UAN* vista até agora é utilizada para descrever ações físicas, de mais baixo nível de abstração, associadas com os dispositivos de interação. É possível construir, sobre essas ações, níveis mais altos de abstração para representar a estrutura completa de tarefas para toda uma aplicação. Nesta seção, são descritas algumas características da notação que possibilitam essa especificação de tarefas num nível mais alto de abstração, enfatizando *relações temporais* entre tarefas[11]. Essas relações são caracterizadas formalmente em [Hartson92b].

Uma relação temporal combina grupos de ações do usuário ou tarefas em uma única tarefa maior. Isto significa que quando um conjunto de ações é conectado com uma relação temporal, uma outra tarefa é criada. A preocupação em especificar as relações temporais entre as tarefas nas fases iniciais de projeto força os projetistas a pensarem sobre os problemas de temporização bem antes da implementação da interface.

[11] O estilo de interação por manipulação direta aumenta a utilização de relações temporais entre tarefas.

A seguir, é apresentada cada uma das relações temporais suportadas pela *UAN* (considerando as tarefas A e B):

- **Seqüência**: $A\ B$

  Consiste do mais simples relacionamento temporal entre tarefas; uma tarefa é executada somente depois que a anterior a ela for completamente terminada. Na *UAN*, uma seqüência é representada escrevendo-se as ações uma após a outra (na horizontal ou vertical). No exemplo apresentado na Seção 4.3, as tarefas em alto nível (*expandir aplicação* e *apagar arquivo*) são feitas em seqüência, pois a segunda só pode ser executada depois da primeira ter sido concluída. As tarefas no nível articulatório são executadas geralmente em seqüência; por exemplo, na movimentação de um objeto na tela têm-se as seguintes ações executadas uma após a outra: mover o cursor para o objeto, pressionar o botão esquerdo do mouse, arrastar a representação do objeto e liberar o botão do mouse.

- **Escolhas**: $A \mid B$

  A relação de *escolha* representa uma disjunção (*OR*). É utilizada para descrever um conjunto de caminhos alternativos para executar uma tarefa, onde o usuário pode escolher somente um por vez. Exemplo: escolha das funções principais de uma aplicação. Combinando *escolhas* com os símbolos de repetição * e +, obtêm-se as chamadas *escolhas repetidas*, representadas por $(A \mid B)^*$ ou $(A \mid B)^+$. Os símbolos de parênteses são utilizados no agrupamento de tarefas. Esta relação é bastante comum na maioria das aplicações, ocorrendo em situações onde o usuário pode executar uma entre várias tarefas apresentadas, repetidamente. No exemplo da Seção 4.3 as tarefas de *apagar arquivo* e *renomear arquivo* foram colocadas numa estrutura de escolhas repetidas (ver descrição da interação utilizando *UAN*, no nível de tarefas).

- **Independência na ordem de execução**: $A\ \&\ B$

  Na utilização de sistemas interativos é comum encontrar situações onde um determinado número de tarefas devem ser executadas, mas a ordem de execução não é importante. No *FManager*, as tarefas de apagar arquivo e renomear arquivo possuem ordem de execução independente quando realizadas em arquivos diferentes. Um outro exemplo clássico de independência na ordem de execução de tarefas é o preenchimento de campos em formulários.

- **Interrupção**: $A \rightarrow B$

  Existem várias maneiras de uma tarefa poder ser interrompida por outra; esta interrupção pode ocorrer devido a ações iniciadas pelo usuário, ou como resultado de ações

do sistema. $A \rightarrow B$ indica que a tarefa B pode ser interrompida pela tarefa A. O exemplo mais comum de utilização desse mecanismo é a ativação do *Help* num sistema, em qualquer parte da interação. Em alguns casos, o projetista necessita definir tarefas que não podem ser interrompidas; na *UAN*, utiliza-se o símbolo <> delimitando estas tarefas. Ações primitivas do usuário geralmente são consideradas como não interruptíveis.

- **Intercalação**: $A \leftrightarrow B$

  Duas tarefas são intercaláveis se uma pode interromper a outra, e vice-versa. Na prática, isto significa que o usuário pode fazer parte de uma tarefa, mudar para outra e depois retornar para a primeira no ponto onde parou e assim por diante. Um exemplo de intercalação bastante utilizado atualmente é a ativação de várias janelas, rodando diferentes aplicações, em sistemas baseados em janelas.

- **Concorrência**: A || B

  Com o uso de intercalação, o usuário pode alternar entre tarefas, mas somente uma tarefa pode estar sendo executada num determinado instante de tempo. A *concorrência* permite que duas tarefas sejam executadas simultaneamente. Esse paralelismo não tem sido muito explorado nas interfaces com o usuário, pois geralmente é difícil obtê-lo numa mesma aplicação, a não ser que o usuário dispare um processo, que continua em execução enquanto ele interage em outra parte da aplicação. Um outro tipo de concorrência é observado nas ações de dois ou mais usuários num *CSCW (Computer-Supported Cooperative Work)*. Esses usuários, utilizando diferentes estações de trabalho, podem executar ações simultaneamente em instâncias compartilhadas da aplicação.

- **Espera**: $A\ (t>n)\ B$

  Intervalos de tempo também são importantes na descrição de tarefas; o projetista pode definir um intervalo de tempo mínimo t entre a execução de duas tarefas. Por exemplo, uma interação onde os resultados de uma tarefa A vão ser utilizados na execução da tarefa B, e a tarefa A leva n segundos para ser completada, pode ser representada em *UAN* por `Tarefa A (t>n) Tarefa B`. No nível articulatório também pode ser explorada essa relação temporal. Por exemplo, na tarefa 1 do exemplo apresentado (Seção 4.3), pode-se definir a velocidade para o clique duplo no botão do mouse, $M\vee\wedge$`(t<n)` $M\vee\wedge$. Essa expressão significa que entre um clique e outro pode ocorrer um tempo de espera máximo de n unidades de tempo.

Resumindo, as tarefas em *UAN* podem ser combinadas da seguinte forma:

- As ações físicas nos dispositivos são tarefas. Exemplos: $\sim$[X] e $M_L \vee \wedge$;
- Se A é uma tarefa, então também são tarefas (A), A, A^*, A^+ e $<A>$;
- Se A e B são tarefas, também o são $A\ B$, $A \mid B$, $A \,\&\, B$, $A \rightarrow B$, $A \leftrightarrow B$ e $A \parallel B$.

Assim, uma descrição de tarefa utilizando *UAN* corresponde a um conjunto de ações, possivelmente agrupadas entre parênteses e separadas por operadores lógicos e temporais. A cada tarefa assim formada pode então ser dado um nome, que é utilizado como referência àquela tarefa. O nome da tarefa usado como referência é chamado de *macro da tarefa* devido ao nome poder ser usado como substituto para toda a descrição da tarefa. O nome de referência pode então ser usado como uma ação do usuário em um outro nível de abstração maior, como em uma chamada de procedimento num programa. Uma grande vantagem de *empacotar* descrições de tarefas em macros é a futura *reutilização* de especificações e obtenção de consistência da interface.

4.6 Simbologia da UAN

Os símbolos da notação *UAN* foram escolhidos com alguns propósitos específicos:

- A utilização do símbolo possui uma *localidade de definição*, similar àquela encontrada em linguagens de programação. Isto ajuda na manutenção da consistência da especificação, pois uma modificação na definição de um método serve para uma classe de objetos. Como exemplo podem ser citados o símbolo de movimento do cursor ($\sim$) e o símbolo que representa feedback na interface (!).
- A *UAN* consiste de uma linguagem textual, todos os símbolos podem ser digitados a partir de um teclado padrão.
- Os símbolos foram escolhidos com o objetivo de serem visualmente onomatopéicos. Por exemplo, $\sim$ sugere movimento, [X] sugere uma caixa em volta de X representando o contexto do objeto, o sinal de exclamação (!) sugere algo que chama atenção, > reflete a noção de acompanhar, enquanto >> sugere o acompanhamento com mudança de tamanho (*rubberbanding*).

As Tabelas 4.1, 4.2 e 4.3 resumem os principais símbolos utilizados pela *UAN*.

O que está sendo representado	**Símbolo UAN**	**Significado**
Objetos e Contextos	X	objeto X (significado muito geral, por exemplo, ícone de arquivo)
	X'	objeto X específico
	[X]	contexto do objeto X, através do qual ele é manipulado
	$[X]_{region}$	contexto do objeto X, descrevendo a *região* através da qual ele é manipulado numa determinada operação
Movimento do cursor	~	move o cursor
	~[X]	move o cursor para o contexto do objeto X
	[X]~	move o cursor para fora do contexto de X
	~[x,y]	move o cursor para uma posição arbitrária x,y
	~[x,y]*	move o cursor zero ou mais vezes para o ponto x,y
	~[x',y']	move o cursor para o ponto específico x',y'
	~[x,y in A]	move o cursor para um ponto arbitrário dentro do objeto A
	~[X in Y]	move o objeto X para dentro do objeto Y
	[X]~	move o cursor para fora do contexto do objeto X
Dispositivos e ações primitivas do usuário	∨	pressionar (*depress*)
	∧	liberar (*release*)
	X∨	pressionar X
	X∧	liberar X
	K"abc"	ler o *string* "abc", via teclado
	K(x)	ler um valor para a variável x, via teclado
Repetição de tarefas	A^*	tarefa *A* é executada zero ou mais vezes
	A^+	tarefa *A* é executada uma ou mais vezes
	A^n	tarefa *A* é executada exatamente n vezes
Condição de viabilidade	*A : B*	*se A então B*
Mudança de intenção	*A; B*	tarefa B não precisa necessariamente ser executada

Tabela 4.1: *Símbolos UAN para descrição de ações e tarefas do usuário no nível articulatório*

O que está sendo representado	**Símbolo UAN**	**Significado**
Agrupamento	( )	mecanismo de agrupamento de tarefas
Seqüenciamento	$A\ B$	tarefas A e B devem ser executadas na ordem da esquerda para a direita, ou de cima para baixo
Escolha	$A \mid B$	escolha de tarefas (usada para mostrar tarefas alternativas)
Escolha repetida	$(A \mid B)^*$	escolha de A ou B, seguida pela escolha de A ou B, zero ou mais vezes.
Independência na ordem de execução	$A\&B$	tarefas A e B são independentes na ordem de execução
Interrupção	$A \rightarrow B$	tarefa A pode interromper a tarefa B
Não interrupção	$< A >$	tarefa A não pode ser interrompida
Intercalação	$A \leftrightarrow B$	tarefas A e B podem ser intercaladas no tempo
Concorrência	$A \parallel B$	tarefas A e B podem ser executadas simultaneamente
Espera	$A(t > n)B$	tarefa B é executada depois de uma espera de mais de n unidades de tempo seguidos da tarefa A

Tabela 4.2: *Símbolos UAN para descrição de ações e tarefas do usuário em níveis mais altos de abstração*

O que é representado	**Símbolo UAN**	**Significado**
Localização	@x',y'	na localização no ponto específico x', y' da tela (por exemplo, para apresentar o objeto X)
	@X	no objeto X
	@x',y' in X	no ponto x',y', do objeto X
Highlight	!	símbolo geral para destacar um objeto
Unhighlight	-!	retira o destaque de um objeto
Blinking	!-!	efeito de *blink* (pisca) em um objeto
Outline	*outline*(X)	apresenta o contorno do objeto X na tela
Display	*display*(X)	apresenta o objeto X na tela
Erase	*erase*(X)	apaga o objeto X da tela
Redisplay	*redisplay*(X)	apaga X e mostra X novamente (em nova posição da tela)
Dragging	X >~	objeto X segue (é arrastado pelo) cursor
Rubberbanding	X >>~	objeto X muda de tamanho enquanto segue o cursor
Expressões para feedback num conj. de objetos	$\forall$, $\neq$, =	mesmo significado matemático

Tabela 4.3: *Símbolos UAN para representação do feedback da interface*

4.7 Complementando as descrições da interface

A *UAN* é uma técnica efetiva para descrever tarefas do usuário numa interface, com informação do feedback, estados e conexões com a aplicação. Entretanto, ela não descreve *layout* gráfico de telas nem informações de transição de estados (que geralmente está distribuída entre as várias descrições de tarefa na coluna de *Estados da Interface*, sem oferecer uma visão macroscópica). A utilização da *UAN* com as características apresentadas não é suficiente para descrever de maneira efetiva uma interface[12]. Para tornar as descrições *UAN* mais completas deve-se complementar as descrições apresentadas com outras representações como telas, cenários, diagramas de transição de estados e tarefas, e comentários de projeto.

4.7.1 Cenários

Devido à *UAN* não apresentar graficamente a aparência das telas e dos objetos da interface, é importante adicionar à notação algumas figuras representativas, que podem ser indicadas como referências nas colunas de *feedback da interface* e anexadas aos documentos de especificação. Às vezes, uma figura é muito mais representativa que uma descrição textual para alguns tipos de informação sobre layout de telas. Por exemplo, na descrição de tarefas do gerenciador de arquivos hipotético da Seção 4.3, são feitas algumas referências a figuras, na coluna de descrição do feedback da interface (`figuras x, y, z`). Nessas figuras devem ser esboçados detalhes de como será o feedback esperado na interface.

4.7.2 Diagramas de estados

Apesar da tabela de especificação *UAN* ter uma coluna explícita para descrição dos estados da interface, não é suficiente para representar uma visão geral dos estados e das transições que ocorrem. Assim, alguns *DTEs* (Seção 3.4.3) podem ser anexados às descrições para complementar a notação, ajudando na futura construção da interface.

4.7.3 Diagramas de tarefas

Como foi apresentado, a *UAN* contém um mecanismo (;) para especificar pontos numa tarefa onde mudanças de intenção do usuário podem ocorrer. Às vezes é importante especificar não somente os pontos onde a tarefa pode ser interrompida, mas o que o usuário

[12] De forma análoga, na análise de sistemas, a utilização única de *diagramas de fluxo de dados*, sem o *dicionário de dados* e outras ferramentas, é insuficiente para descrição de um sistema.

pode fazer para interromper a tarefa. Esses detalhes devem ser apresentados num nível mais alto, separados da descrição de ações e tarefas – um nível que mostre os relacionamentos entre essas tarefas. Uma forma de se fazer isso é através dos *Diagramas de Transição de Tarefas* (*DTTs*) [Siochi91], que mostram graficamente[13] um conjunto de tarefas relacionadas que o usuário pode executar no sistema.

4.7.4 Discussões de projeto

Devido à representação de projeto ser um documento de trabalho, servindo de comunicação entre várias equipes, os projetistas devem incluir comentários e razões que os levaram a tomar as decisões no projeto da interação, que compromissos estavam envolvidos, quais as possíveis vantagens e desvantagens na decisão tomada. Em um processo de desenvolvimento iterativo envolvendo várias pessoas, é importante saber por que uma determinada característica de projeto foi utilizada, e por que outras foram descartadas.

4.7.5 Colunas adicionais

Em [Siochi91] são apresentadas algumas sugestões de extensão das colunas da tabela de especificação *UAN*. Algumas das informações que essas colunas-extra podem incluir são:

- Ações na memória do usuário, carregamento de informação na memória, por exemplo, funções como *RecallLongTermMemory(items)* que indica a recuperação de um item da memória de longa duração.
- Ações perceptuais e cognitivas do usuário, por exemplo, *LookAt(location)*, *LookFor(item)*. Essas ações cognitivas podem ser vistas como instruções dadas ao usuário engajado numa tarefa, antes dele executar as ações físicas.
- Objetivos e intenções do usuário, conforme [Norman86b], para especificar a finalidade da tarefa.
- Numeração das tarefas, para facilitar referência cruzada em grandes projetos.

[13] A notação para os *Diagramas de Transição de Tarefas* é semelhante à notação para os *DTEs* (nodos, arcos, e símbolo de fechamento de tarefa).

4.8 Poder analítico da UAN

Embora a motivação básica da *UAN* seja simplesmente fornecer uma notação prática para representar interação em interfaces de manipulação direta, a abstração obtida pelas descrições pode ajudar os projetistas a detectarem ambigüidades e inconsistências em projetos de interface. Espera-se que depois de algum tempo trabalhando com descrições *UAN*, o projetista esteja apto a descobrir mais facilmente possíveis problemas no projeto. Esse fato se deve principalmente à concisão e precisão das especificações *UAN*; as descrições prosaicas, ou mesmo descrições como *GOMS*, são difíceis de analisar, principalmente devido ao volume da documentação produzida.

Em *UAN*, a utilização de referências a sub-tarefas em descrições de tarefas fornece um meio direto de agrupamento das ações do usuário. Por exemplo, a seqüência $M_R\vee\ M_R\wedge$ é assimilada como um clique simples do botão direito do mouse $M_R\vee\wedge$. Depois de algum tempo utilizando a notação, o projetista começa a fazer esse tipo de agrupamento em níveis mais altos. Por exemplo, o projetista pode assimilar a seqüência $\sim$[X]$M_L\vee\wedge$ como uma ação atômica para seleção de um objeto X. Dessa forma, muitas ações comuns, como por exemplo, invocar um menu *pulldown*, fazer uma escolha num menu, movimentar um objeto, logo começam a tornar-se automáticas para a pessoa que está lendo (ou escrevendo) as especificações *UAN*. Essa característica ajuda na investigação de possíveis erros na interface.

4.9 Contexto de desenvolvimento para a UAN

Para utilizar a *UAN* efetivamente, os desenvolvedores da interface devem ter em mente o contexto onde ela melhor se encaixa no processo de desenvolvimento. Para isto, foram considerados o *ciclo de vida estrela* (Seção 2.5) e os seis focos do espaço de decisão em projeto de interfaces, do *modelo snowflake* (Seção 2.6.1).

4.9.1 UAN e o ciclo de vida estrela

No ciclo de vida estrela (Figura 2.8), a *UAN* pode ser utilizada primeiramente na fase de projeto conceitual e representação de projeto. Na conceituação do projeto o projetista faz esboços da aparência e comportamento da futura interface. A *UAN* pode ser utilizada para captura rápida das características iniciais da interface e para estudar o projeto à medida que ele evolui.

Também é possível utilizar a *UAN* na fase de análise de tarefas e análise funcional. A *UAN* serve como representação tanto para a estrutura hierárquica de tarefas produzida durante a análise de tarefas como para seqüências de ações primitivas produzidas durante o projeto de baixo nível. A principal diferença entre os dois tipos de análise é o nível de abstração – descrições de tarefas *UAN* em níveis mais altos são definidas por seqüências temporais de descrições *UAN* nos níveis mais baixos.

Por ser uma representação da interface no domínio comportamental (visão do usuário), as descrições *UAN* devem ser traduzidas para o domínio de construção equivalente, antes ou durante o processo de prototipação.

Na Figura 2.8, a ligação entre a fase de projeto da interface e a fase de avaliação (central) é suportada pela *UAN*, principalmente devido às descrições serem compactas e possuírem um alto grau de precisão, permitindo uma revisão cuidadosa do projeto para encontrar possíveis inconsistências (ver Seção 4.8), ou simplesmente verificar a conformidade do que foi especificado com o que foi implementado pela equipe de programadores.

4.9.2 UAN e o modelo snowflake

As decisões tomadas em cada um dos focos do modelo *snowflake* (Figura 2.9) devem ser capturadas, servindo de meio de comunicação entre os projetistas, que podem estar produzindo diferentes partes da interface.

No foco de *atividades mentais do usuário* a *UAN* pode ser utilizada como um meio de representação da estrutura de tarefas num nível mais alto. O foco de *atividades físicas do usuário* é fortemente suportado pela *UAN* na coluna de especificação de ações do usuário das tabelas de especificação. A *UAN* suporta o *projeto funcional da interface* em termos comportamentais, isto é, as ações que um usuário executa para realizar uma tarefa e o feedback e as mudanças de estado causadas por essas ações. Um suporte completo para esse foco requer a tradução da representação comportamental da interface para uma representação no domínio de construção. O foco de *projeto físico da interface* é suportado pela coluna de feedback das especificações *UAN*, juntamente com a utilização de cenários que ilustram telas, janelas e vários outros objetos da interface. Uma vez que as decisões tomadas no foco de *políticas de implementação* afetam diretamente as decisões de projeto, os seus efeitos podem ser capturados nas notas e discussões para cada descrição de tarefa.

4.10 Resumo

Neste capítulo foi apresentado o básico do formalismo *UAN* para especificação de projetos de interação homem-computador, e um exemplo de sua aplicação. A notação foi analisada dentro de modelos que contemplam o desenvolvimento de sistemas interativos. O capítulo não cobre todas as características da *UAN*, mas o que foi fornecido é suficiente para entendimento de grande parte das descrições que utilizam essa notação.

No capítulo seguinte é apresentada uma aplicação dessa notação na descrição de uma interface na área de modelagem geométrica. Com base nessa aplicação, faz-se uma análise do poder de expressão da notação, apresentam-se alguns problemas encontrados com a sua utilização e sugerem-se algumas extensões que poderiam melhorá-la.

Capítulo 5

Aplicação, Propostas de Extensão e Avaliação do formalismo UAN

Uma linguagem, qualquer linguagem, é um meio de comunicação, e deve ser julgada exclusivamente como tal.
Luís Fernando Veríssimo

Neste capítulo discute-se sobre a aplicação da notação *UAN* na descrição de parte da interface do *ProSIm*[1], no processo de síntese de imagens. São feitas críticas analíticas à notação e propostas algumas alterações para melhorar a sua utilização.

5.1 Uma interface gráfica para o ProSIm

Resumidamente[2], o *ProSIm* (Prototipação e Síntese de Imagens Fotorealistas e Animação) consiste num conjunto de módulos que implementam funcionalidades para síntese de imagens. O projeto iniciou-se em 1988 e vem sendo coordenado pelo Professor Léo Pini Magalhães e a Professora Wu, Shin-Ting do Departamento de Computação da FEE/Unicamp.

Diversos módulos foram criados e gradualmente desenvolvidos, de forma que os principais métodos de síntese de imagens foram abordados (modelagem geométrica, geração de texturas, animação e visualização). Recentemente, constatou-se a necessidade de facilitar a utilização dos módulos por pessoas que não estivessem diretamente envolvidas com o projeto. O sistema deveria então ser provido de uma interface gráfica, através da qual todos os

[1] Uma breve descrição deste sistema é fornecida na Seção 5.1.
[2] Uma descrição mais detalhada do *ProSIm* pode ser encontrada em [Malheiros94].

módulos pudessem ser acessados e as escolhas dos diversos parâmetros fosse feita de forma interativa, fornecendo uma realimentação visual adequada para o usuário.

Esta interface vem sendo desenvolvida [Malheiros93, Malheiros94] na linguagem C, para rodar no ambiente *UNIX* com o sistema de janelas *XWindow*, também conhecido como *X11*. Na construção da interface também foi utilizada a *toolkit XView* que segue o padrão *Open Look* da *AT&T* e *Sun Microsystems*.

Conforme [Malheiros93], o objetivo inicial na construção de tal interface seria definir um *layout* para os diversos tipos de interação, eventualmente servindo de ponto de partida para um trabalho mais aprofundado e certamente mais complexo de estruturação de uma interface definitiva.

5.2 Aplicação da UAN

Tendo o objetivo principal do trabalho se restringido à aplicação e avaliação da notação *UAN*, resolveu-se utilizá-la para especificar a interface do *ProSIm*, e a partir dessa experiência propor alterações que pudessem melhorar o poder de expressão da notação. É importante ressaltar que o objetivo do trabalho realizado era utilizar o desenvolvimento da interface do *ProSIm* como "bancada de testes" para a avaliação da *UAN*. Entretanto, uma interface para o processo de síntese de imagens como um todo pode ser bastante complexa, devido ao grande número de interações existentes. Para a avaliação que pretendia-se realizar, foi considerado suficiente utilizar a *UAN* para descrever somente parte da interface do *ProSIm*, principalmente aquela relacionada com a fase de Modelagem Geométrica.

Convém ressaltar que no ciclo de vida estrela (Seção 2.5, Figura 2.8) normalmente a fase de implementação da interface é realizada depois que a equipe de desenvolvimento possui alguma especificação da mesma. Como parte da interface do *ProSIm* já estava implementada quando começou-se a trabalhar com o Grupo de Computação e Imagens (GCI), foi resolvido, numa primeira etapa do trabalho, especificar parte da interface que já estava implementada. Descobriu-se que essa abordagem é bastante satisfatória para se fazer avaliação de interfaces, pois durante as sessões de especificação, várias críticas foram feitas e algumas inconsistências descobertas. A tarefa de especificar a interface levou à discussão de várias características que muitas vezes são deixadas de lado.

Outra observação importante é que a interface foi especificada da maneira como achava-se que ela *deveria* ser. Assim, as especificações apresentadas não traduzem exatamente

a interface como ela está disponível hoje pois, durante a especificação, já foram incluídas as alterações consideradas interessantes para melhorar a sua usabilidade. Algumas dessas alterações dizem respeito até mesmo à seqüência em que as diversas fases do processo de síntese de imagens deveriam ser apresentadas a um usuário inexperiente. Algumas dessas sugestões foram bem-aceitas pelo GCI, mas outras ainda consistem em pontos de discussão. Isso já era esperado, devido à natureza subjetiva e intuitiva do projeto de interfaces.

5.3 Propostas de Extensão no formalismo UAN

No Apêndice A apresenta-se parte das especificações *UAN* realizadas para a interface de modelagem geométrica. Mais especificamente, concentrou-se na descrição da interface referente à interação do usuário com o ambiente 3D via um *mouse* comum (para maiores detalhes sobre este trabalho ver [Velasquez95]). O motivo de ter se escolhido este sub-domínio do problema justifica-se pelos crescentes esforços que têm sido empreendidos para se obter uma interação amigável entre objetos tridimensionais projetados em telas de visualização e o usuário. Muitos trabalhos têm se voltado nesta direção, procurando resolver este problema de comunicação. Sendo assim, investiu-se na especificação de ferramentas de interação e suporte visual que facilitem a interação com o espaço tridimensional, principalmente durante a fase de modelagem geométrica no processo de síntese de imagens.

A partir da experiência obtida durante a especificação da interface foram sugeridas algumas extensões[3] à notação. Essas extensões tem por objetivo aumentar o poder de expressão da *UAN*, ou simplesmente facilitar a sua utilização. A seguir, será discutida sucintamente cada uma dessas propostas.

5.3.1 Condições de Viabilidade

A *UAN*, da forma como foi apresentada no Capítulo 4, não permitia especificar ações condicionais do tipo *se então senão*. A notação foi estendida com o comando:

[3]Nota-se que as especificações no Apêndice A possuem várias características não apresentadas na notação básica *UAN*, descrita no Capítulo 4.

conds ? *açõesdoentão* : *açõesdosenão*

onde *conds* pode ser um conjunto de condições, separadas por E e OU. As ações do *então* e do *senão* consistem num conjunto de tarefas, podendo até mesmo ser outros comandos de condição.

Para a descrição de algumas interações também decidiu-se incorporar à notação o comando *case* para especificar vários conjuntos de condições, da mesma forma que as linguagens usuais de programação. Os símbolos { e } foram acrescentados à linguagem para marcar, respectivamente, o início e o fim do escopo de um determinado comando.

Acredita-se que essas extensões na linguagem para descrição de ações e tarefas aumentam o seu poder de expressão e não comprometem a facilidade de uso e entendimento da mesma, pois esses comandos são considerados simples, mesmo para aqueles que não sejam da área de computação.

5.3.2 Tratamento de Exceções

De uma maneira geral, a *UAN* não especifica o que o usuário *pode* fazer numa determinada interação, mas sim o que ele *deve* fazer, o procedimento correto, mais simples, para realizar determinada tarefa[4]. Dessa forma, os programadores da interface ficam livres para tomar decisões a respeito de políticas a serem tomadas quando o usuário não executa determinada tarefa da forma como seria esperado. Isso não é desejável, pois a interface final pode apresentar inconsistências.

As *condições de viabilidade* podem ser usadas pelos projetistas para ajudar no tratamento de exceções e prevenção de possíveis erros cometidos pelo usuário, através da não permissão de tarefas inviáveis. As condições de viabilidade com a cláusula *senão* aumentam o poder de expressão, mas as especificações podem se tornar confusas quando cada ação não esperada do usuário for especificada dentro da cláusula *senão*. Assim, decidiu-se estender a tabela de especificação *UAN* básica (Figura 5.1) com uma coluna adicional para o tratamento de exceções e possíveis erros cometidos pelo usuário durante a execução de determinada tarefa.

As *tabelas de especificação estendidas* apresentam então o seguinte *layout*:

[4] Por exemplo, na descrição da tarefa 3, para apagar um arquivo no *FManager* (Seção 4.3), que decisão tomar se o usuário ao movimentar o arquivo para a cesta de lixo resolve abandoná-lo no *workspace*?

Tarefa: <*nome-da-tarefa*>				
Ações do Usuário	**Feedback da Interface**	**Exceções**	**Estados da Interface**	**Conexão com Aplicação**
⋮	⋮	⋮	⋮	⋮

Figura 5.1: *Layout da tabela de especificação UAN estendida*

5.3.3 Manipulação no espaço 3D

Em [Silva93] é comentado que uma das maiores limitações numa interface de modelagem geométrica está na não existência de uma forma sistemática de definição de entidades geométricas no espaço tridimensional. Na interface do *ProSIm* foi incorporado um novo componente de interface 3D: uma área de desenho tridimensional, mais especificamente um *canvas* tridimensional [Velasquez95]. Assim, o novo componente tridimensional da interface exigia uma notação específica para a sua descrição. Estendeu-se a notação *UAN* de forma que ela permitisse, ao nível de ações do usuário, não só uma descrição de interação bidimensional (para as áreas de desenho como o *canvas* 2D, e objetos como *scrollbars* e menus), como também uma descrição das posições espaciais dessa nova componente tridimensional. É importante ressaltar que essa interação 3D é feita através de um dispositivo comum 2D, o mouse.

5.3.4 Feedback Textual

O gênio do cinema-suspense, Alfred Hitchcock, certa vez fez o seguinte comentário:

Para mim, um dos principais pecados de um roteirista, ao encontrar dificuldades, é dizer "Podemos encobrir isso com uma linha de diálogo". O diálogo deve ser um som entre outros sons, apenas algo que sai da boca de pessoas cujos olhos contam a história em termos visuais.

Infelizmente, em se tratando de interfaces de computador, nem sempre só o feedback visual é suficiente. Por exemplo, mudar visualmente a aparência da cesta de lixo para indicar

que um arquivo vai ser apagado pode passar desapercebido pelo usuário, sendo nesse caso preferível mandar uma mensagem textual (ou sonora) explícita dizendo que o arquivo será removido.

A princípio a idéia era acrescentar uma coluna a mais nas tabelas de especificação *UAN* para separar a descrição do feedback visual (semântico) do feedback textual (sintático). Entretanto, para evitar um aumento excessivo no número de colunas da tabela, optou-se por criar uma função *FbkTxt(Mens)* que associa às ações do usuário, o feedback textual que deve ser dado ao usuário durante a interação. Uma tabela com o texto para cada mensagem deve ser acrescida à documentação gerada pelas especificações da interface. A mensagem fornecida ao usuário deve ser concisa e explicativa, e de preferência escrita pelo pessoal de *fatores humanos*[5].

A posição na interface onde será exibida a mensagem ao usuário deve ser, de preferência, fixa, mantendo-se assim a consistência da interface. Essas posições devem ser determinadas durante o projeto do *layout* da interface. O sucesso de algumas interfaces deve-se em grande parte à clareza e consistência desse tipo de feedback fornecido ao usuário. Em interfaces de manipulação direta deve-se fornecer um feedback constante para as ações associadas aos botões do mouse numa região pré-definida da tela.

5.3.5 Formato do Cursor

Um outro tipo de feedback que deve ser explícito durante a especificação da interface é aquele referente ao formato do cursor para diferentes ações do usuário. Para facilitar a especificação desse formato, na coluna de feedback da interface, criou-se uma função *Muda-Cursor2D(Tipo)* e acrescentou-se às descrições uma tabela para os tipos de cursor utilizados.

5.3.6 Descrições em alto nível

Geralmente, as colunas da tabela de especificação *UAN* para feedback associado às ações do usuário, estados da interface e conexão com a aplicação são utilizadas somente nos níveis mais baixos de especificação (nível articulatório). Assim, optou-se por não fazer as descrições *UAN*, ao nível de tarefas, na forma tabular. A idéia é descrever a interação através de uma linguagem de uso geral, utilizando os operadores temporais discutidos na Seção 4.5, comandos de condição e repetição, e operadores para agrupamento de tarefas e comandos.

[5]Existem vários guias para feedback textual, por exemplo, a mensagem *Programa abortado* não é recomendada para mulheres que estejam grávidas.

5.3.7 Comentários nas tabelas

Uma característica adicional nas especificações apresentadas no Apêndice A é a utilização de comentários dentro da tabela (delimitados pela seqüência de símbolos //). Da forma como a *UAN* foi definida no Capítulo 4 não ficava claro se algumas linhas da especificação consistiam de comentários de projeto ou descrições de tarefas e ações propriamente ditas. Esses comentários numa forma mais explícita podem ser úteis para os programadores da interface na documentação dos programas.

5.3.8 Dispositivos

A *UAN* não sugere nenhuma padronização para os símbolos utilizados na representação do teclado[6]. Nas especificações realizadas utiliza-se uma possível padronização para as teclas comumente utilizadas; por exemplo, *KOpen, KEsc, KEnter, KCut, KPaste.* Essa associação de símbolos para o teclado está relacionada com a plataforma das estações *SUN*, onde a interface do *ProSIm* vem sendo desenvolvida atualmente.

5.3.9 Indexação da Tabela

Uma dificuldade comum quando da especificação da interface, foi a constante referência que era feita às células da tabela, para discussão de uma determinada característica da interação. De forma a facilitar essas referências, as células foram indexadas, como índices de uma matriz, como pode ser observado nas tabelas de especificação apresentadas no Apêndice A.

5.3.10 Símbolos adicionais

Em [Hix93] os autores comentam que a notação *UAN* é aberta, podendo a equipe de desenvolvimento da interface se sentir livre para definir os seus próprios símbolos, ou modificar os já existentes, conforme as necessidades encontradas durante a representação do projeto. Durante a especificação da interface do *ProSIm* foram definidos, ou modificados, alguns símbolos, comentados a seguir. A necessidade desses símbolos adicionais está relacionada com a necessidade de que todos os símbolos da notação possam ser digitados a partir de um teclado padrão.

[6] A tendência das interfaces atuais é utilizarem cada vez menos o teclado, sendo a interação sendo quase toda feita através do *mouse* ou outro dispositivo que exija menos esforço do usuário.

- Para efeito de simplicidade, o símbolo para clique duplo foi definido pela letra **w** após o nome do dispositivo. Assim, um clique duplo no botão esquerdo do mouse seria representado por `MLw` ao invés de `MLv^v^`.
- Como foi definido na *UAN* básica que todos os símbolos poderiam ser digitados a partir de um teclado padrão[7] (Seção 4.6), o quantificador universal $\forall$ foi representado pelo símbolo `pt` (paratodo).
- A repetição de uma tarefa um número n de vezes era representada na *UAN* básica por $(Tarefa)^n$. Na *UAN* estendida essa repetição é descrita por $(Tarefa) * n$.
- Na *UAN* básica, o contexto de um objeto é representado pelos símbolos de colchetes envolvendo o nome do objeto. Por exemplo, para mover para uma das extremidades de uma linha, representar-se-ia por $\sim[linha]_{extremidade}$. Na *UAN* estendida, sugere-se que os colchetes envolvam a parte manipulável do objeto; assim, a mesma tarefa seria representada por `~[linha.extremidade]`.

5.3.11 Entrada de dados acrescida de sintaxe

Para alguns tipos de entrada de dados textual (por exemplo, nome de um arquivo, data e hora correntes) pode ser interessante acrescentar a sintaxe às descrições. Percebe-se então a necessidade de manutenção de uma gramática para descrever a sintaxe formal de alguns comandos de entrada numa determinada aplicação. Esta gramática poderia ser definida à parte, com as descrições *UAN* apenas referenciando os símbolos não-terminais da mesma. Dessa forma, simplificam-se as descrições pois, a princípio, detalhes dessa natureza são importantes apenas para o programador da interface.

5.3.12 Tabela de Variáveis e Funções

Com o objetivo de facilitar a leitura e entendimento das tabelas de especificação, é anexada às descrições uma outra tabela com as principais variáveis e funções utilizadas nas descrições da interface. Nesta tabela são especificadas algumas informações como nomes, onde/como são utilizadas e descrição do conteúdo das variáveis e funções[8]. Dessa forma, as tabelas de especificação não ficam muito carregadas desse tipo de informação, que geralmente só interessa aos programadores ou a quem precisa analisar mais detalhadamente a interface.

[7] A idéia de continuar com essa restrição está relacionada com uma futura construção de um compilador para a linguagem.

[8] Isso lembra um *dicionário de dados* utilizado para descrever o conteúdo de alguns objetos definidos durante a *análise estruturada* [Pressman92].

5.4 Documentação da fase de especificação da interface

Partindo-se da notação básica *UAN*, e acrescentando as propostas discutidas na Seção 5.3, sugere-se que a documentação final gerada pela fase de especificação da interface seja basicamente aquela representada pela Figura 5.2.

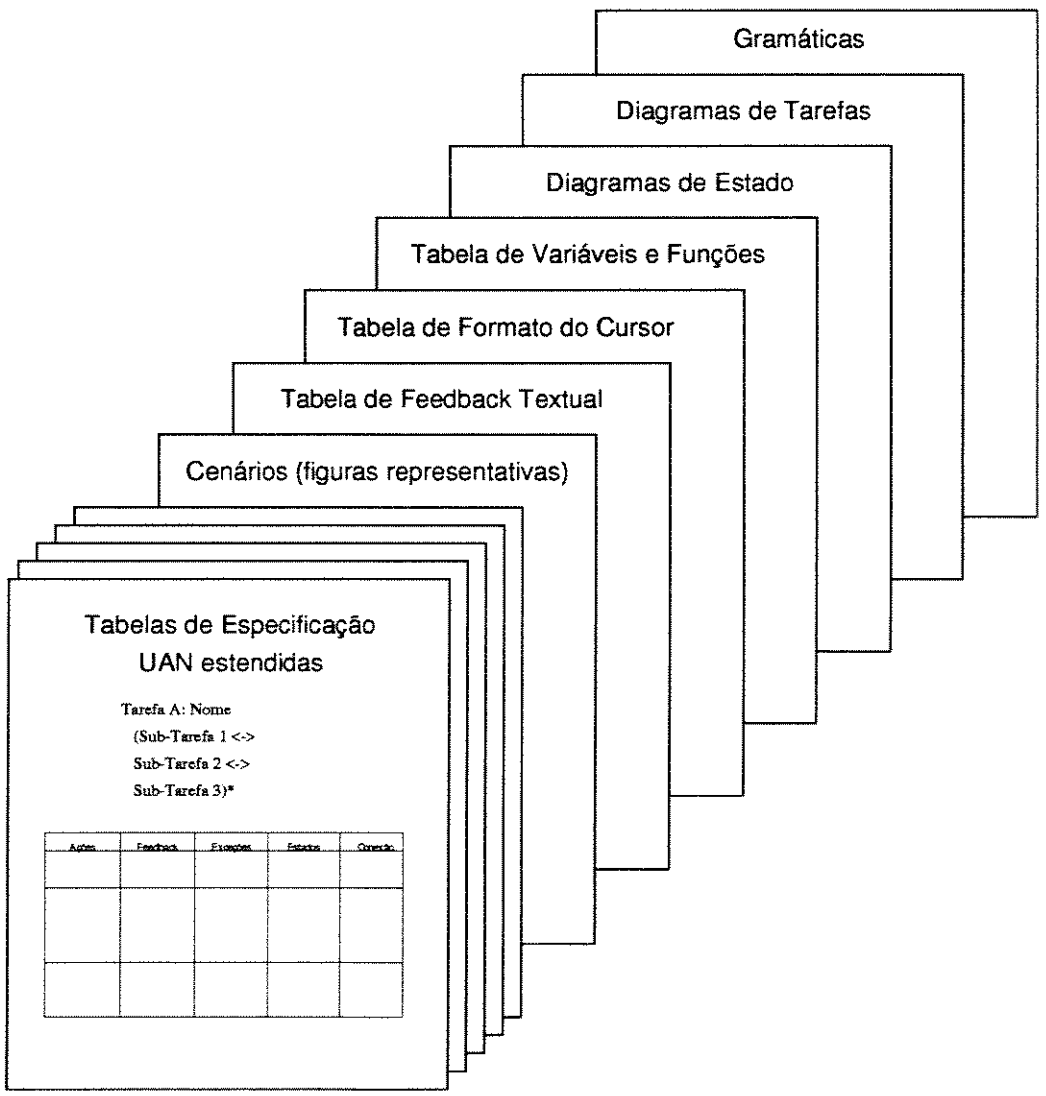

Figura 5.2: *Documentação final produzida pela fase de especificação da interface*

5.5 Avaliação da UAN

Na Seção 5.3, apresentam-se algumas críticas e sugestões com relação à notação *UAN*, baseada na experiência obtida durante a especificação de parte da interface do *ProSIm*. Algumas dessas sugestões serviram somente para adequar a notação à plataforma de desenvolvimento da interface[9].

Nesta seção, é apresentada, em linhas gerais, uma avaliação da notação *UAN*, sem necessariamente propor soluções para os problemas identificados. Essa avaliação é baseada na experiência adquirida com a utilização da notação.

5.5.1 Vantagens

A primeira característica atrativa da *UAN* é que ela fornece um conjunto de operadores para especificar relações temporais entre tarefas. Em particular, os operadores de intercalação, paralelismo e espera entre tarefas são melhorias consideráveis em relação às notações E/OU usadas nas linguagens de representação de tarefas convencionais.

A segunda vantagem da notação é a forma explícita como ela representa os estados e feedback do sistema a partir das ações do usuário. O formalismo utilizado para isto (isto é, as colunas da tabela de especificação) fornecem ao projetista um *layout* claro e natural para checar ligações lógicas entre entrada, saída e os estados internos do sistema relevantes para a interface.

Uma terceira característica interessante na notação pode ser considerada comum à maioria das notações formais. Uma descrição formal da interface, como *UAN* tende a ser, abre um caminho para a automatização dos testes de usabilidade. Por exemplo, através do exercício de especificação da interface do *ProSIm*, identificaram-se algumas regras que poderiam ser incorporadas a uma ferramenta de teste de usabilidade baseada em *UAN*. Algumas dessas regras são descritas a seguir:

- Se, para seqüências idênticas de ações do usuário, o feedback da interface ocorre de maneiras diferentes, então o feedback do sistema pode não estar consistente.
- Se o feedback de um objeto é mapeado em diferentes posições da tela, o *layout* do sistema pode estar inconsistente.

[9]Se estivesse sendo especificada, por exemplo, uma *interface multimodal*, ter-se-ia que adequar a notação aos diversos dispositivos utilizados; e as ações do usuário seriam bem diferentes daquelas que normalmente ele executa numa interface convencional de manipulação direta.

- Se na descrição de uma tarefa for identificado um número grande de ações do usuário, essa tarefa deve ser reestruturada dentro da interface. Por exemplo, na descrição da interface inicial do *ProSIm*, foram identificadas cinco ações primitivas do usuário para instanciar um objeto na cena. Após uma reestruturação, essa tarefa consistia apenas de duas ações primitivas.
- As ações do usuário que não possuem feedback associado devem ser reanalisadas. A ausência de feedback inibe a utilização do sistema por usuários inexperientes.

Outras características interessantes na notação:

- Facilita a comunicação entre os membros da equipe de desenvolvimento da interface.
- Possui poucos símbolos, é fácil de memorizar.
- Produz um volume reduzido de documentação. Por exemplo, a documentação da interface para modelagem geométrica no *ProSIm* totalizou cerca de 20 páginas. Não foi feita uma descrição prosaica detalhada da interface para se fazer uma comparação do volume de especificação produzido mas, conforme [Hix93], a documentação gerada pelas descrições *UAN* podem reduzir em até cinco vezes a documentação final da interface.

5.5.2 Limitações

As limitações identificadas na *UAN* são principalmente referentes à falta de uma semântica clara e algumas deficiências no seu poder de expressão.

A falta de uma semântica clara está relacionada com as relações temporais entre as descrições nas diversas colunas da tabela de especificação. Da forma como foi apresentado, uma descrição da interface deve ser entendida lendo-se a tabela a partir da primeira linha, da esquerda para a direita, varrendo-se todas as linhas. Assim, quando se está lendo uma linha i, as declarações na coluna de ações do usuário devem ser primeiro consideradas, depois a coluna de feedback e assim por diante. Depois de um tempo utilizando a notação, percebeu-se que esse *algoritmo* nem sempre se aplica para interações mais complexas. Torna-se necessária uma notação que expresse o escopo e relacionamentos, tais como paralelismo ou seqüenciamento, entre as colunas da tabela.

Em [Hix93] é comentado que algumas pessoas acham que as descrições *UAN* são muito detalhadas para serem usadas nas primeiras etapas de projeto, podendo limitar a criatividade do projetista. Outros comentam que os símbolos da notação poderiam ser mais expressivos.

Uma outra crítica em relação à notação é a maneira como é representado o feedback da interface. Na *UAN* básica o destaque em um objeto (!) não é representado explicitamente. Por um lado é interessante poder especificar a interface sem se preocupar se esse destaque será, por exemplo, uma mudança de cor, ou de estilo, de uma determinada linha. Entretanto, esses detalhes não devem ser deixados para que o programador decida o que fazer. Na especificação da interface do *ProSIm*, detalharam-se alguns tipos de feedback a serem fornecidos, na coluna de conexão com a aplicação, mas talvez fosse melhor adicionar uma nova coluna à tabela de especificação para representar esse tipo de informação.

5.6 Resumo

Neste capítulo foram propostas algumas extensões na notação básica *UAN* apresentada no Capítulo 4. Essas propostas originaram-se a partir da utilização da *UAN* para especificar parte da interface de um sistema de síntese de imagens, o *ProSIm*, discutido brevemente no início deste capítulo. Também apresentou-se uma avaliação informal da notação, baseando-se na experiência adquirida com a sua utilização.

Capítulo 6

Ferramentas de Software para suporte à UAN

O homem é limitado pelas ferramentas que utiliza.
Dijkstra

A utilização de qualquer técnica de especificação de interface pode ser bastante melhorada através da utilização de uma ferramenta automática que apóie essa tarefa. Neste capítulo descreve-se sucintamente a ferramenta automática *QUANTUM*, utilizada durante a especificação da interface do *ProSIm*.

6.1 Uma ferramenta automática para UAN

Uma ferramenta de suporte automático para a *UAN* consistiria basicamente de um editor para a tabela de especificação, através do qual os projetistas pudessem entrar com as descrições das tarefas do usuário. Neste sentido, a ferramenta seria similar a uma planilha eletrônica com simples edição de texto dentro de cada célula, podendo ser acrescida a facilidade de definir colunas adicionais. A ferramenta também poderia ter facilidades de edição gráfica (desenho das telas, cenários e diagramas de estado e tarefas), e *hipertexto* para ajudar o projetista a automatizar referências a outras descrições, notas e figuras.

Estabeleceu-se desde o princípio do trabalho que a obtenção de uma ferramenta automática seria muito importante na fase de especificação da interface do *ProSIm*. Entretanto, não foi traçado como objetivo principal deste trabalho a implementação de tal ferramenta; o principal interesse estava centrado na notação *UAN* em si, sua aplicação e validação.

Através de uma lista de discussões sobre *HCI*, na rede *internet*, foi estabelecido um contato com um pesquisador nos Estados Unidos (no Departamento de Ciência da Computação da Universidade de *Virginia*), que esteve trabalhando na implementação de uma ferramenta baseada em *UAN* para especificação de interfaces. Esta ferramenta foi enviada para o Grupo de Interfaces do DCA, inclusive os programas-fonte. As restrições impostas caso houvesse alterações na ferramenta foram relativas à não comercialização e manutenção do nome da instituição.

6.2 A ferramenta QUANTUM

QUANTUM (*Quick User Action Notation Tool for User-Interface Management*) é um ambiente que permite aos projetistas da interface a criação e manipulação de especificações de projeto utilizando *UAN*. Trata-se de um protótipo escrito na linguagem *Tcl/Tk*. No Apêndice B são descritos alguns detalhes sobre essa linguagem e apresentados alguns trechos de programa que implementam a ferramenta *QUANTUM*.

Algumas adaptações e extensões propostas na notação *UAN* (discutidas no Capítulo 5) foram introduzidas à ferramenta *QUANTUM*, sendo que hoje as principais características apresentadas por ela são:

- Facilidade de criação e manutenção de vários projetos de interface.
- Especificação visual da hierarquia de tarefas do usuário (*Abstraction Hierarchy Window*).
- Manutenção de palheta de cores para agrupamento de tarefas relacionadas.
- Fácil manipulação (inserção, eliminação e movimentação) na estrutura da rede de tarefas.
- Manutenção de tabela com sintaxe de algumas descrições básicas em *UAN*, facilitando a especificação de tarefas.
- Manutenção de tabela de referência entre sintaxe e semântica, com exemplos de alguns símbolos *UAN*.
- Edição fácil nas *tabelas de especificação UAN*.

6.3 Utilização básica da ferramenta QUANTUM

Ao ser ativada, a ferramenta *QUANTUM* apresenta a tela mostrada na Figura 6.1.

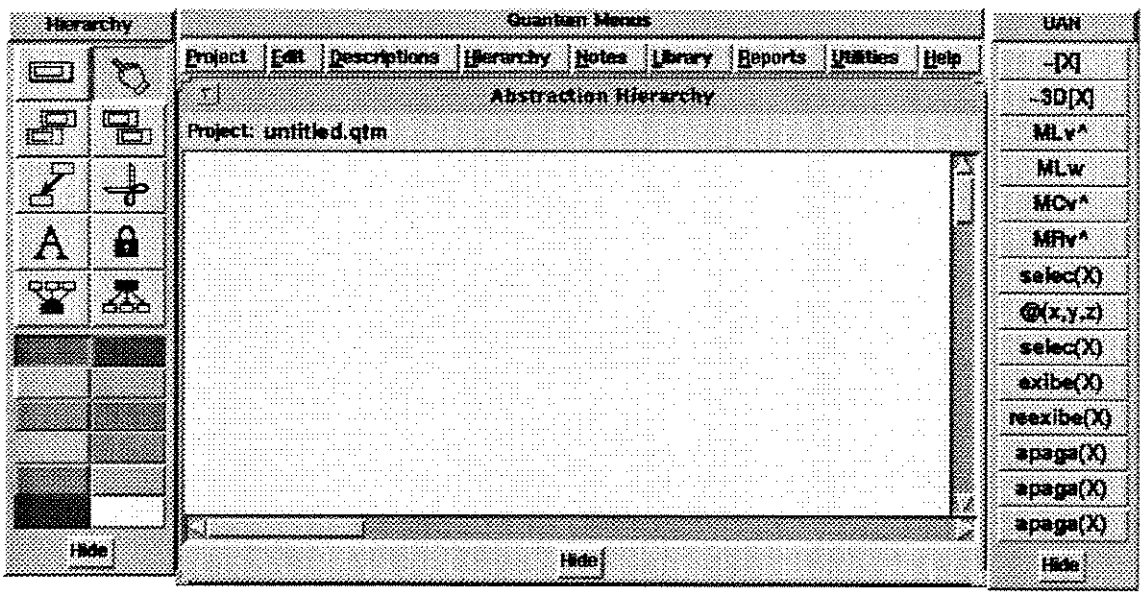

Figura 6.1: *Tela principal da ferramenta QUANTUM*

A interação na ferramenta *QUANTUM* é feita basicamente da seguinte forma:

1. Editar rede de tarefas (*Abstraction Hierarchy*) da interface.

   Esta é uma fase geralmente *top-down*. *QUANTUM* gera um arquivo texto (*.qtm*) com formato próprio. O projetista pode subdividir o projeto da interface em vários desses arquivos, para cada nível de descrição de tarefas. Um exemplo mostrando o formato desse arquivo é apresentado no Apêndice C. As ações comuns do usuário na edição de uma rede de tarefas são:

   - adicionar, copiar, apagar, editar, mover e renomear tarefas;
   - generalizar/especificar tarefas.

   A maioria das opções pode ser realizada por manipulação direta na janela de abstração, estando também disponíveis via menus (*QUANTUM* menus), ou através do uso de

teclas aceleradoras. Na Figura 6.2 apresenta-se uma possível rede de tarefas para a interface do *ProSIm* em nível mais alto de abstração.

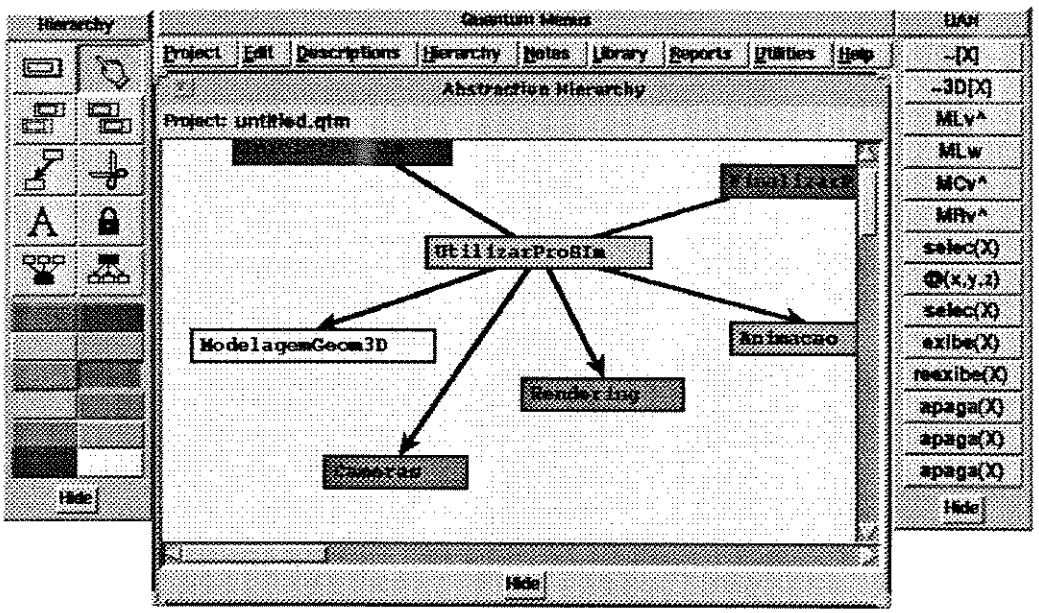

Figura 6.2: *Um exemplo de rede de tarefas descrita em QUANTUM*

2. Especificar as ações do usuário para cada tarefa identificada na interface.

A partir da rede de tarefas descrita anteriormente, o projetista pode selecionar tarefas específicas (clique duplo sobre a tarefa) para fazer a descrição *UAN*. É apresentada uma tabela de descrição de tarefas (baseada nas tabelas estendidas de especificação *UAN*), onde o projetista descreve a interação em termos de ações do usuário, feedback da interface, tratamento de exceções, estados da interface e conexões com a computação. É importante ressaltar que a ferramenta não faz ainda nenhum tipo de checagem em relação à sintaxe das descrições. Na Figura 6.3 é mostrado um exemplo de uma tabela de descrição de tarefas apresentada pela *QUANTUM*:

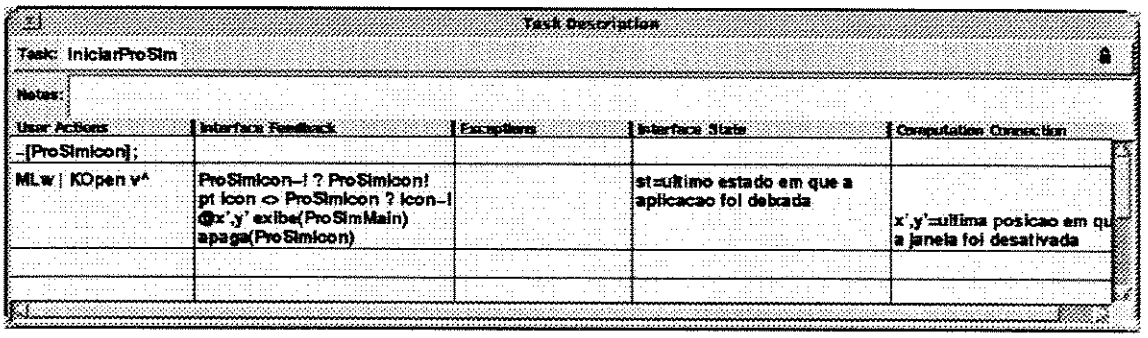

Figura 6.3: *Tabela de especificação UAN apresentada pela ferramenta QUANTUM*

A ferramenta possui outras características interessantes, por exemplo, um sistema de ajuda ao usuário (Figura 6.4) bastante detalhado, *popups* para criar novos projetos, abrir e salvar projetos existentes, *fan-in* e *fan-out* na rede de tarefas.

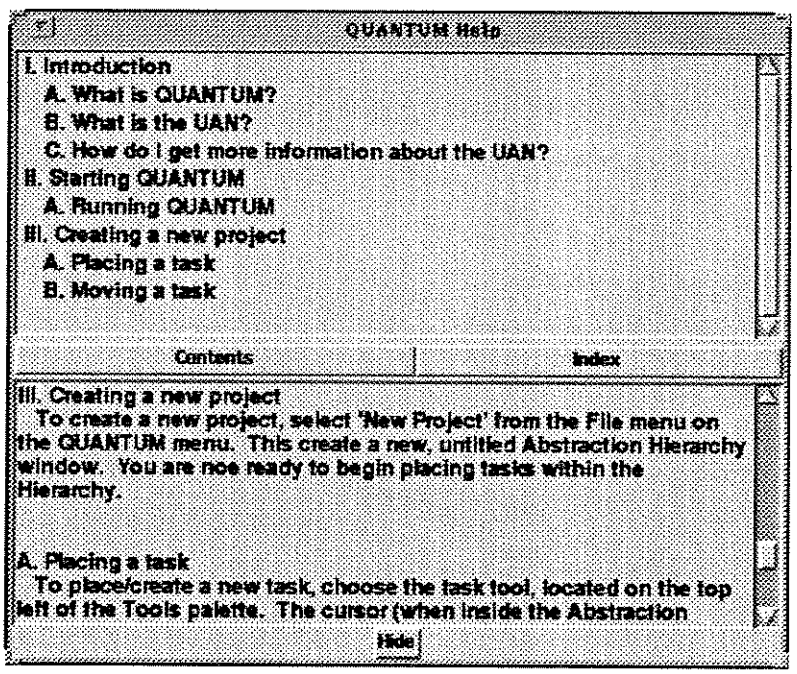

Figura 6.4: *Sistema de Ajuda para a Ferramenta QUANTUM*

6.4 Resumo

Neste capítulo foi comentada a importância de utilização de uma ferramenta automática durante a fase de especificação do projeto de uma interface. Apresentou-se sucintamente a ferramenta *QUANTUM*, a forma como ela foi obtida, e a interação básica numa sessão de uso da ferramenta. Maiores detalhes sobre a ferramenta, a linguagem em que foi implementada e os arquivos que são gerados com a sua utilização encontram-se nos Apêndices B e C.

Capítulo 7

Conclusões e Sugestões para Trabalhos Futuros

Um artista nunca termina de fato
o seu trabalho, ele apenas o abandona.
Paul Valéry

A pesquisa no campo de *Interação Homem-Computador* tem se firmado como uma das áreas de maior interesse de grandes empresas de software espalhadas pelo mundo. O objetivo final da maioria dos trabalhos desenvolvidos na área é tornar a interação com o sistema mais confortável e mais eficiente. Muitos pesquisadores têm investigado formas de se desenvolver interfaces que garantam a aceitação do produto por parte da comunidade usuária. Devido à natureza multidisciplinar da área, essas formas de desenvolvimento visam, principalmente, permitir que pessoas que não possuam conhecimento aprofundado de computação e engenharia de software sejam capazes de projetar boas interfaces.

Neste trabalho foram analisadas uma variedade de técnicas para *comunicar* o projeto da interface entre os membros da equipe de desenvolvimento. Enfatizaram-se as principais características em cada uma dessas técnicas, comparando-as e apontando vantagens e limitações na sua utilização. Uma análise mais criteriosa, baseada na utilização de cada uma dessas técnicas deve ser, posteriormente, considerada.

O formalismo *UAN* (*User Action Notation*) foi escolhido como a notação a ser utilizada nas descrições de interface a serem realizadas pelo grupo. Algumas das razões para esta escolha é que *UAN* é simples, concisa, centrada nas ações e tarefas do usuário (*domínio comportamental*) e apropriada para descrever interfaces de manipulação direta. Essa notação foi discutida detalhadamente (sintaxe e semântica) no Capítulo 4.

Com o objetivo de verificar, e validar, o poder de expressão do formalismo *UAN* no projeto de interfaces, decidiu-se aplicá-la na descrição da interface para modelagem geométrica do sistema *ProSIm*. Buscou-se identificar, durante a especificação da interface, limitações que a notação apresenta e, ao mesmo tempo, propor soluções para elas. Teve-se também a oportunidade de constatar muitas das vantagens apontadas na notação.

O que chamou mais atenção na experiência de utilização da *UAN* foi que, apesar de não ser totalmente formal, ela pode indicar possíveis inconsistências na interface. Uma descrição textual da interação seria muito propensa a erros de interpretação. Outra característica importante é que a sua flexibilidade na formalização da interface permite que profissionais de diferentes áreas possam entendê-la e utilizá-la de forma eficiente. As principais limitações da notação se referem à falta de uma semântica clara e algumas deficiências no seu poder de expressão. Para muitas dessas deficiências foram sugeridas extensões que poderiam ser acrescentadas à notação, facilitando a sua utilização e aumentando o seu poder de expressão. Uma utilização mais exaustiva da notação, com outros exemplos e circunstâncias a que se aplica, poderá validar as extensões propostas.

Concluído o trabalho, acredita-se que a *UAN* é uma boa técnica para especificação de projeto de interfaces, sendo recomendada tanto para especificação de interfaces existentes (auxiliando na tarefa de avaliação) como para interfaces a serem criadas.

7.1 Contribuições do Trabalho

Ao final de três anos de pesquisa (dois deles dedicados exclusivamente à area de *Interação Homem-Computador*), as principais contribuições alcançadas foram:

- ▷ Estudo da área de representação em projeto de interfaces, onde foi apresentada uma classificação para as principais técnicas e uma análise informal das mesmas, apontando os seus principais problemas e vantagens de sua utilização.

- ▷ Estudo detalhado da notação *UAN* e sua aplicação no desenvolvimento de uma interface não-trivial. Foram sugeridas algumas propostas de extensão na notação, com o objetivo de aumentar o seu poder de expressão e facilitar a sua utilização.

- ▷ Instalação da ferramenta automática *QUANTUM* para especificação de interfaces. Algumas alterações foram feitas nessa ferramenta de forma a incluir algumas das extensões propostas neste trabalho.

- natureza interdisciplinar do trabalho, envolvendo diferentes áreas de conhecimento (Engenharia de Software, Computação Gráfica, Artes e Lingüística).

7.2 Trabalhos Futuros

Em termos de continuidade do trabalho, são sugeridos os seguintes desdobramentos:

- Obtenção de uma descrição formal mais rigorosa para a linguagem *UAN* (por exemplo, através de uma gramática *BNF*). Um pré-requisito para aplicação de uma *avaliação analítica* mais completa das descrições *UAN*é que esta deveria ter uma definição de gramática mais rigorosa (por exemplo, regras de produção do tipo *BNF*), pois esse tipo de análise geralmente requer processamento automático das descrições de tarefa. Entretanto, existe um compromisso a ser considerado: se a avaliação analítica é facilitada, certamente as descrições vão se tornar mais difíceis de entender.

- Construção de um compilador para a linguagem *UAN*, com geração automática de código para uma *IDL*[1] específica.

- Pensar em *UAN* como uma linguagem visual de comunicação, com alto poder de expressão. *Linguagens visuais* para descrição de ações do usuário têm sido estudadas por vários pesquisadores, mas muitos problemas ainda precisam ser resolvidos com as linguagens de especificação textuais.

- Fazer uma avaliação empírica da *UAN* estendida, através de um método estatístico estruturado para verificar se as extensões propostas neste trabalho consistem realmente em melhorias na notação.

- Formalização das demais etapas do processo de síntese de imagens para o *ProSIm* (módulos de câmeras, *rendering* e animação).

- Aperfeiçoamento da ferramenta *QUANTUM* para especificação de interfaces. Ainda existem muitas características que precisam ser implementadas para facilitar a utilização da ferramenta *QUANTUM*, de forma que ela possa ser plenamente explorada durante a fase de projeto de interfaces.

[1] Interface Development Language.

- Dependendo das ferramentas que estejam sendo utilizadas para a construção da interface, muitas das características não precisam ser especificadas em alto nível de detalhes. Por exemplo, na implementação da interface do *ProSIm* foi utilizada a *toolkit XView*, que já possui muitos dos estilos de interação (aparência e comportamento) padronizados pelo *OpenLook*. Um trabalho interessante seria descrever, em *UAN*, os diversos estilos de interação de diferentes padrões disponíveis no mercado, como o *Motif* e o *OpenLook*, e fazer uma comparação entre eles, ao nível de ações do usuário.

- Investigação do teste de interfaces descritas em *UAN*. Como qualquer outra atividade que envolve a participação do homem, o processo de desenvolvimento de interfaces também está sujeito à inserção de defeitos. Pretende-se investigar a aplicação da Teoria do Teste de Software, já estabelecida para o teste de software em geral, no teste de interfaces.

- Investigar o poder de expressão da *UAN* na especificação de interfaces para ferramentas de teste de software. Alguns estudos já foram realizados [Vilela94a, Vilela94b], buscando identificar características necessárias para tornar uma ferramenta de teste de software mais efetiva no seu objetivo de conduzir a atividade de teste com sucesso.

- Expandir a utilização da *UAN*. Atualmente, a notação vem sendo utilizada na especificação da interface de um sistema de informação na área de saúde, o SIS, em desenvolvimento na Universidade Federal de Ouro Preto - MG, com resultados bastante satisfatórios. Várias inconsistências têm sido descobertas antes que a interface seja definitivamente implementada. A interface do sistema segue o padrão comercial, mostrando o poder da *UAN* para descrever também tais tipos de interface.

- Os sistemas interativos tornam-se mais *amigáveis* à medida em que os usuários podem trabalhar da mesma maneira em que trabalham no mundo real. Geralmente, os usuários trabalham cooperando uns com os outros na realização de tarefas complexas e inter-relacionadas. A área de pesquisa em *Sistemas Interativos Distribuídos* envolve o estudo de sistemas computacionais que suportem grupos de pessoas engajadas numa tarefa comum, fornecendo uma interface para esse ambiente compartilhado. Um trabalho interessante seria investigar uma possível extensão na notação *UAN* para especificar o trabalho cooperativo.

7.3 Considerações Finais

Sabe-se que a atividade de projeto, embora devesse englobar grande parte de um ciclo de desenvolvimento, geralmente é deixada para trás. Especificar o projeto de uma interface é uma tarefa difícil, cara e pode ser até mesmo enfadonha; mas é indiscutivelmente necessária se se deseja obter uma interface final com alta usabilidade.

Foi constatado que ainda existe uma lacuna muito grande entre as abordagens convencionais de desenvolvimento de software e o desenvolvimento de interfaces com o usuário. Investimentos maciços em pesquisa precisam ser realizados no campo de projeto e avaliação de sistemas interativos, de forma a organizar e estruturar esse processo. Espera-se que alguma contribuição tenha sido dada nesse sentido.

Acredita-se que os objetivos estabelecidos no início deste trabalho foram alcançados e que os resultados obtidos aqui fornecem subsídios suficientes para trabalhos futuros que venham a implementar, validar e estender o que foi apresentado.

Apêndice A

Especificação da interface do ProSIm

A seguir, apresenta-se parte da documentação gerada pelas especificações *UAN* realizadas para a interface de modelagem geométrica do sistema *ProSIm*.

A.1 Descrição da Rede de Tarefas

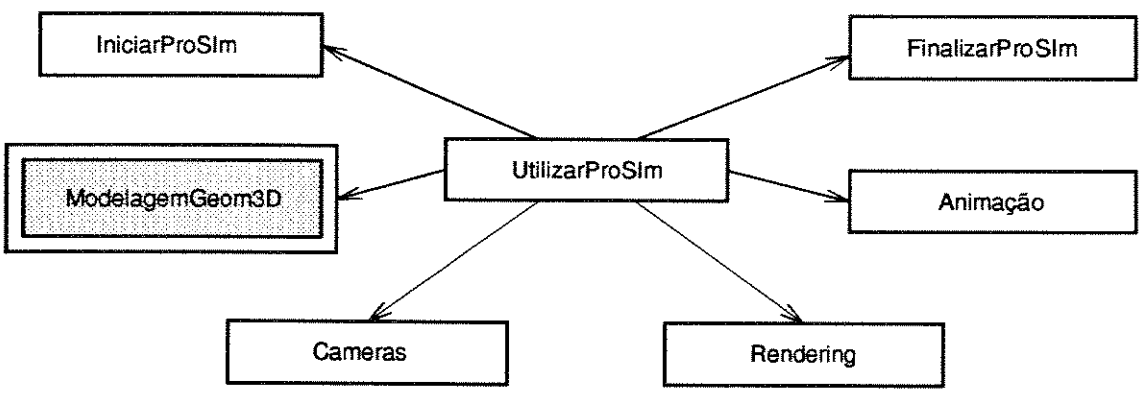

Figura A.1: *Rede de tarefas para o ProSIm em alto nível*

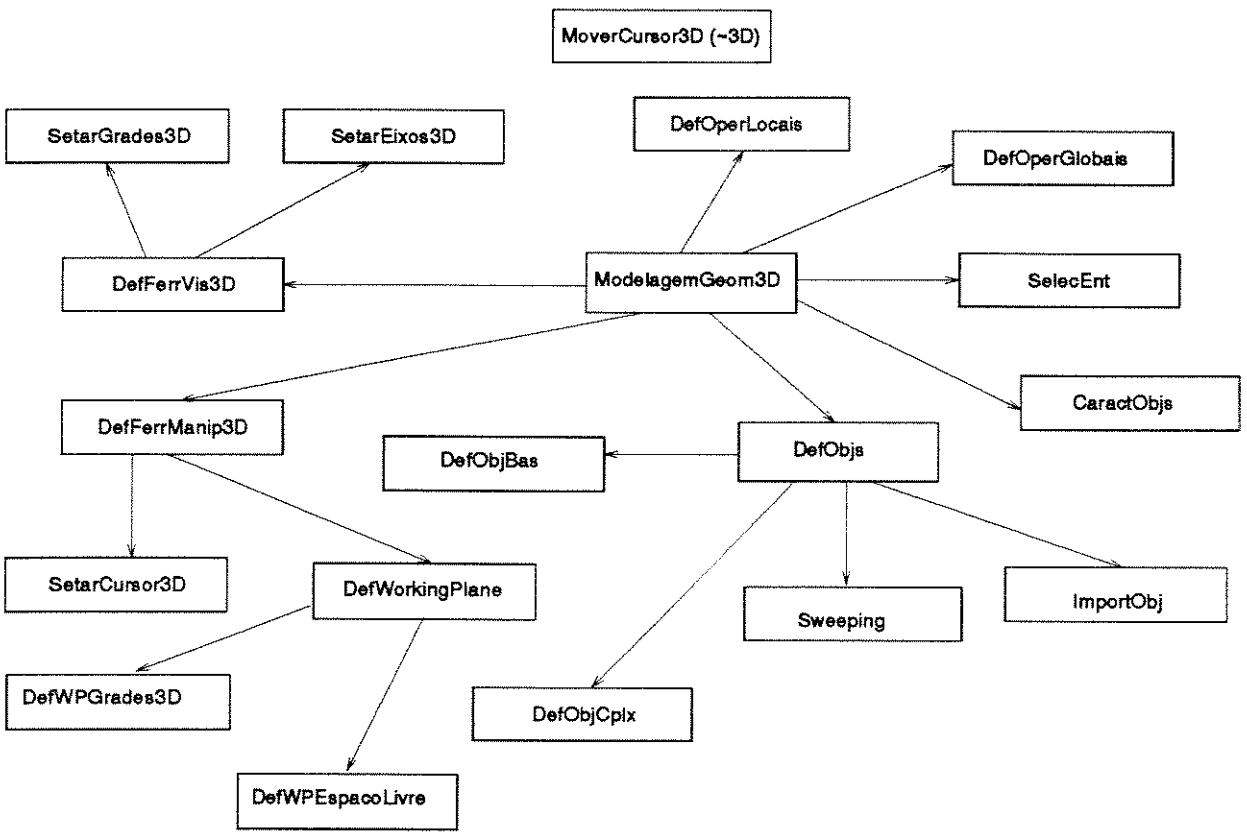

Figura A.2: *Rede de tarefas para a fase de Modelagem Geométrica*

A.2 Tabelas de Especificação

Tarefa: *UtilizarProSIm*

// descricao de mais alto nivel de uma sessao de uso do ProSIm //

IniciarProSIm

(Modelagem Geométrico 3D <->

Manipulação de Câmeras <->

Renderering da cena <->

*Produção da Animação)**

FinalizarProSIm

	Tarefa: *IniciarProSIm*									#1
	Ações do Usuário		**Feedback**		**Exceções**	**Estados**		**Conexões**		
1	~[ProSImIcon];	1		2	3		4			5
2	ML w | KOpen v^	1	ProSImIcon-! ? ProSImIcon! pt Icon <> ProSImIcon ? Icon-! @x',y' exibe(ProSImMain) apaga(ProSImIcon)	2	3	st=ultimo estado em que a aplicacao foi deixada	4	x',y' = ultima posicao em que a janela foi desativada		5

	Tarefa: *FinalizarProSIm*									#6
	Ações do Usuário		**Feedback**		**Exceções**	**Estados**		**Conexões**		
1	~[QuitItem in ProSImMenu];	1		2	3		4			5
2	MLv^	1	@x', y' exibe(ConfWindow) (Conf==Sim) ? { apaga(ConfWindow) apaga(ProSImWindow) } : apaga(ConfWindow)	2	3	st= estado final	4	pt w, ProSImWindow(w) ? { SalvaAmbiente(w) apaga(w) }		5

Tarefa: *Modelagem Geometrica 3D*

DefFerrVis3D

DefFerrManip3D

DefObjs ?

{ (Selec(EntSelec))

SelecEnt

(Entidade Selecionada = Object|Component) ?

DefOperGlobais : DefOperLocais

DefCaractObjs }

Tarefa: *DefFerrVis3D*

// Definir Ferramentas Visuais 3D //

(SetarGrades3D |

*SetarEixos3D)**

Tarefa: *DefFerrManip3D*

// Definir Ferramentas de Manipulacao 3D //

((SetarCursor3D ?

*DefWorkingPlane)**

Tarefa: *DefWorkingPlane*

(DefWPGrades3D |

*DefWPEspacoLivre)**

Tarefa: *SetarGrades3D* // habilita ou desabilita o recurso visual de grades 3D no espaco de cena //				#2.1.1
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1 selec(Grades3D)	2 (Grades3D=off) ? @[cena] exibe(Grades3D)): @[cena] apaga(Grades3D) // Grade3D são três planos ortogonais. Cada plano possui marcações formando uma rede quadriculada. As grades 3D podem ser apresentadas em projeção perspectiva ou paralela, dependendo da opção do usuário (a projeção default é a perspectiva) (ver figura 1) //	3	4 st=Grades3DOn st=Grades3DOff	5 // selec atraves de um menu toggle //

Tarefa: *SetarEixos3D* // habilita ou desabilita o recurso visual de eixos 3D no espaco de cena //				#2.1.2
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1 selec(Eixos3D)	2 (Eixos3D=off) ? @[cena] exibe(Eixos3D)): @[cena] apaga(Eixos3D) // Os eixos 3D estão centralizados na origem das coordenadas reais. Constituem-se de três retas pontilhadas, rotuladas com x,y,z, respectivamente ortogonais entre si, que se interceptam no mesmo ponto (ver fig. 2) //	3	4 st=Eixos3DOn st=Eixos3DOff	5 // selec atraves de um menu toggle //

Tarefa: *SetarCursor3D* // habilita ou desabilita a manipulacao no espaco tridimensional //				#2.2.1
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1 selec(Cursor3D) 1	2 (Cursor3D=off) ? @[cena] exibe(Cursor3D)}: @[cena] apaga(Cursor3D) // o cursor3D consiste de três retas, rotuladas com x, y, z, respectivamente, ortogonais entre si, que se interceptam no mesmo ponto. O cursor3D é posicionado na cena, de acordo com a posição do observador (ver figura 3) //	3	4 st=Cursor3DOn st=Cursor3DOff	5 // selec atraves de um menu toggle //

Tarefa: *~3D* // Mover cursor 3D no espaço tridimensional //				
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**

1 (~[x,y,z em cena])* 1	2 MudaCursor2D(2)
	Cursor3D >~
	// O cursor 3D acompanha o movimento do cursor 2D //
	case {
	~>[eixoX] ? { eixoY~> eixoZ~> }
	~>[eixoY] ? { eixoX~> eixoZ~> }
	~>[eixoZ] ? { eixoX~> eixoY~> }
	@[FbkXYZ] exibe(x,y,z atuais)
	// feedback numerico, representando a nova posicao do cursor 3D //
		3	4
			st=MovEsp3D
				5
				atualiza x,y,z

2 [x,y,z em cena] ~ 1	2 MudaCursor2D(1) apaga(cursor3D)	3	4 st=MovEsp2D	5

Tarefa: *DefWPGrades3D* // Definir Working Plane sobre um dos planos das grades tridimensionais //				#2.2.2.1
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1 ~[cena];	2 FbkTxt(1) MudaCursor2D(2)	3	4	5
2 ~3D;	2	3	4 st=markG	5
3 @[x',y' em grade'] MLv^	2 grade'-! ? grade'! pt grade <> grade' ? grade-! // marcacao e' mutuamente exclusiva //	3 // se o ponto nao for precisamente determinado, e' atraido para a grade mais proxima //	4	5 // selec por manipulacao direta do plano desejado // // o destaque e' uma cor diferente na grade //

Tarefa: *DefWPEspacoLivre* *// Definir Working Plane numa posicao qualquer do espaco tridimensional //*				#2.2.2.2
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1 ~[cena]	2 FbkTxt(2) MudaCursor2D(3)	3	4	5
2 (~3D	2	3	4 st=markpt	5
3 @x',y',z' MLv^,)3	2 @[xi',yi',zi'] exibe(marca) // i=(1,2,3), os 3 pontos que definem um working plane // (i=3) ? {apaga(marcas) existe(wkp)? reexibe(wkp): exibe(wkp) } // o working plane consiste de um paralelogramo definido por tres pontos P1, P2, P3 (ver figura x) //	3 // caso o usuario nao forneca os tres pontos, e mude de tarefa, ignorar os pontos marcados //	4	5 // marca e' um ponto colorido //

Tarefa: *DefObjs*
// Definir objetos na cena //
(DefObjBasico |
DefObjCplx |
Sweeping |
*ImportObj)**

Tarefa: *DefObjBasico*
// Definir objeto basico (solido, casca, curva, superficie) //
(DefSolidoseCascas |
DefCurvas |
*DefSuperf)**

Tarefa: *DefSolidoseCascas* //Definir solidos e cascas para objetos basicos (cubo, esfera, cilindro e cone) na cena // #2.3.1.1

	Ações do Usuário	Feedback	Exceções	Estados	Conexões
1	selec(ObjBas); 1	FbkTxt(3) 2	3	4	// selec atraves de um menu iconico // 5
2	~[cena] 1	contorno(ObjBas) >~ 2	// caso o usuario mova o cursor para fora da cena, o contorno do objeto deve desaparecer // 3	st=PosicObjBas 4	// optou-se por uma representacao do objeto (contorno) seguindo o cursor, tanto nas operacoes de posicionamento e dimensionamento, pois o esforco computacional e' menor na realimentacao visual
3	~3D 1	MudaCursor2D(3) FbkTxt(4) contorno(ObjBas) >~3D 2	idem 3	4	5
4	@(x',y',z') MLv^ 1	FbkTxt(5) 2	3	st=DimensObjBas 4	5
5	~3D 1	contorno(ObjBas) >>~3D 2	idem 3		5
6	@(x',y',z') (MCv^ | 1	// Confirma o objeto centrado na posicao x', y', z' // @)x',y',z') exibe(ObjBas) 2	3	4	// objeto deve ser adicionado a lista de estruturas // 5
7	MRv^) 1	// Nao confirma o objeto na cena // MudaCursor2D(1) apaga(contorno(ObjBas)) 2	3	4	5

Tarefa: *DefCurvas* // Definir curvas na cena //				#2.3.1.2
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1 selec(Curva); 1	FbkTxt(6) 2	3	4	// selec atraves de um menu iconico // 5
2 selec(TipoCurva); 1	2	3	4	// selec(TipoCurva) atraves de um menu toggle. TipoCurva pode ser, por exemplo, Bezier, BSpline e Hermite. TipoCurva default sera' BSpline //
3 ~[cena] 1	2	3	4	5
4 (~3D 1	2	3	st=PosicPtsCtrl 4	5
5 1 @(x',y',z') MLv^)+ ?	@(x',y'z') exibe(marca) 2	3	4	// marca e' um ponto colorido // 5
6 { ~3D 1	2	3		5
7 @(x',y',z') (MCv^ | 1	// exibe curva baseada nos ptos. definidos (ptos. de controle) // pt x, marca(x) ? apaga(x) exibe(curva) 2	3	4	5
8 MRv^) } 1	// Nao confirma curva na cena // pt x, marca(x) ? apaga(x) 2	3	4	5

Tarefa: *DefSuperf* // Definir superficies na cena //				#2.3.1.3
Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1 selec(Superf); 1	FbkTxt(7) 2	3	4	// selec atraves de um menu iconico // 5
2 selec(TipoSuperf); 1	2	3	4	// selec(TipoSuperf) atraves de um menu toggle. TipoSuperf pode ser, por exemplo, Bezier, BSpline e Coons Bicubica. TipoSuperf default sera' BSpline //
3 ~[cena] 1	2	3	4	5
4 (~3D 1	2	3	st=PosicPtsCtrl 4	5
5 1 @(x',y',z') MLv^)+ ?	@(x',y'z') exibe(marca) 2	3	4	// marca e' um ponto colorido // 5
6 { ~3D 1	2	3		5
7 @(x',y',z') (MCv^ | 1	// exibe superficie baseada nos ptos. definidos (ptos. de controle) // pt x, marca(x) ? apaga(x) exibe(Superf) 2	3	4	5
8 MRv^) } 1	// Nao confirma curva na cena // pt x, marca(x) ? apaga(x) 2	3	4	5

Tarefa: *DefObjCplx*

// Define objetos complexos, a partir de operacoes booleanas em primitivas //

ExecTransfAfim

*(SelecObjPrim;)*2 ?*

{ SelecOperBool;

ConfNovoObj }

	Tarefa: *SelecObjPrim* // Selecionar objeto primitivo na cena //				#2.3.2.2
	Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1	1 ~[cena]	2 FbkTxt(3) MudaCursor2D(2)	3	4	5
2	1 (~3D [ObjPrim'.manip]	2	3	4 st=SlcObjPrim	5
3	1 MLV^)*	2 ObjPrim'-! ? ObjPrim'! : ObjPrim'-!	3	4	5 // destaque do objeto atraves da mudanca de cor. A intersecao dos objetos deve ser destacada com uma cor diferente da cor de selecao //
4	1	2 existe(ObjPrim!) ? intersecao(ObjPrim,ObjPrim')! MudaCursor2D(1)	3	4	5

	Tarefa: *SelecOperBool* // Seleciona operacao booleana (operacao a ser realizada sobre as primitivas selecionadas) //				#2.3.2.3
	Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1	1 (selec(operbool))*	2 exibe(operbool(Obj1,Obj2)!) apaga(Obj1) apaga(Obj2)	3	4	5 // selec(operbool) atraves de um menu iconico // // exibe(operbool(obj1,obj2)!) apresenta na cena o resultado da operacao entre os dois objetos primitivos. O destaque deve ser mudanca de cor no objeto resultante //

	Tarefa: *ConfNovoObj* // Confirma o novo objeto na cena //				#2.3.2.4
	Ações do Usuário	**Feedback**	**Excessões**	**Estados**	**Conexões**
1	1 Conf(NovoObj)	2 exibe(ConfWindow)	3 // Se o usuario nao confirmar, as estruturas antigas sao mantidas //	4	5 // entidades selecionadas sao agrupadas num novo objeto. Objetos primitivos sao retirados da estrutura //

Tarefa: *Sweeping*

// Define um objeto na cena atraves de um sweeping //

(Grades3D = on?

{ DefWPGrades3D

DefPerfilWP

DefWPGrades3D

*DefTrajWP })**

	Tarefa: *DefPerfilWP* // Define um perfil no working plane //					#2.3.3.2
	Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**	
1	(~3D [Wkp'] 1	2	3	4	5	
2	@[x',y', z' em Wkp'] MLv^;)+ ? 1	@ [x',y', z' em Wkp'] exibe (marca) 2	3	4	5	
3	(~3D [Wkp] 1	2	3	4	5	
4	@[x',y',z' em Wkp'] (MCv^ | 1	exibe (perfil) 2 pt x, marca(x) ? apaga(x)	3	4	// o perfil segue a ordem de entrada dos pontos // 5	
5	MRv^) 1	pt x, marca(x) ? apaga(x) 2	3	4	5	

	Tarefa: *DefTrajWP* // Define uma trajetoria no working plane //					#2.3.3.4
	Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**	
1	(~3D [Wkp'] 1	2	3	4	5	
2	@[x',y', z' em Wkp'] MLv^;)+ ? 1	@ [x',y', z' em Wkp'] exibe (marca) 2	3	4	5	
3	(~3D [Wkp] 1	2	3	4	5	
4	@[x',y',z' em Wkp'] (MCv^ | 1	exibe (traj) 2 pt x, marca(x) ? apaga(x) exibe (objeto) // sweep. transl. //	3	4	// a trajetoria segue a ordem de entrada dos pontos // 5	
5	MRv^) 1	pt x, marca(x) ? apaga(x) 2	3	4	5	

Tarefa: *SelecEnt*

// Selecionar entidade na cena //

(Selecionar Objeto /

Selecionar Componente /

Selecionar Vertice /

Selecionar Aresta /

*Selecionar Poligono)**

	Tarefa: *SelecEnt* // Selecionar entidade na cena (objeto, componente, vertice, aresta, poligono) //				#2.5
	Ações do Usuário	**Feedback**	**Exceções**	**Estados**	**Conexões**
1	(~3D [Obj.manip] MLv^ |	Obj-! ? Obj! : Obj-!			
2	~3D [Obj.componente] MLv^)*	Obj.componente-! ? Obj.componente! : Obj.componente-!			
3	~3D [Obj.vertice] MLv^ |	Obj.vert-! ? Obj.vert! : Obj.vert-!			
4	~3D [Obj.aresta] MLv^ |	Obj.aresta-! ? Obj.aresta! : Obj.aresta-!			
5	~3D [Obj.aresta1] ML v^ ~3D [Obj.aresta2] MLv^ |	Obj.aresta1-! ? Obj.aresta1! : Obj.aresta1-! Obj.poligono-! ? Obj.poligono! : Obj.poligono-!			// o poligono e' definido pelas arestas 1 e 2 //

Tarefa: *DefOperLocais*

// Define um conjunto de operacoes locais que podem ser executadas sobre vertices, arestas e poligonos //

(Duplicar |
Bevel |
Extrude |
ChipOff |
Refinement |
Rounding |
Blending |
Offset |
Remove |
*Export)**

Tarefa: *DefOperGlobais*

// Define um conjunto de operacoes globais que podem ser executadas sobre objetos e componentes //

(Dupliar |
OrientacaoReversa |
TransformacoesAfins |
Deformacao |
Seccionamento |
Smoothing |
Remove |
*Export)**

Tarefa: *CaractObjs*

// Define um conjunto de caracteristicas para os objetos em cena (opcoes complementares) //

(ShowVolume |
ShowArea |
ShowOffset |
ShowNormalVector |
ShowCurvaGeodesica |
Show BoundingBox |
*Show PoligonoControle)**

A.3 Figuras representativas

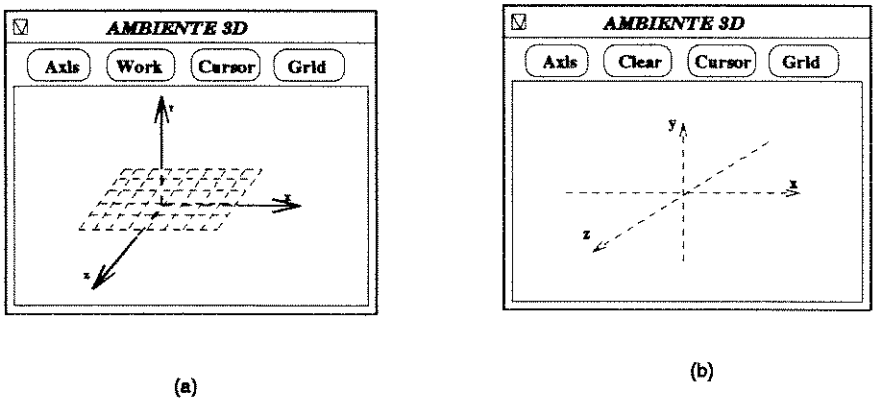

Figura A.3: *(a) Working Plane, (b) Eixos tridimensionais*

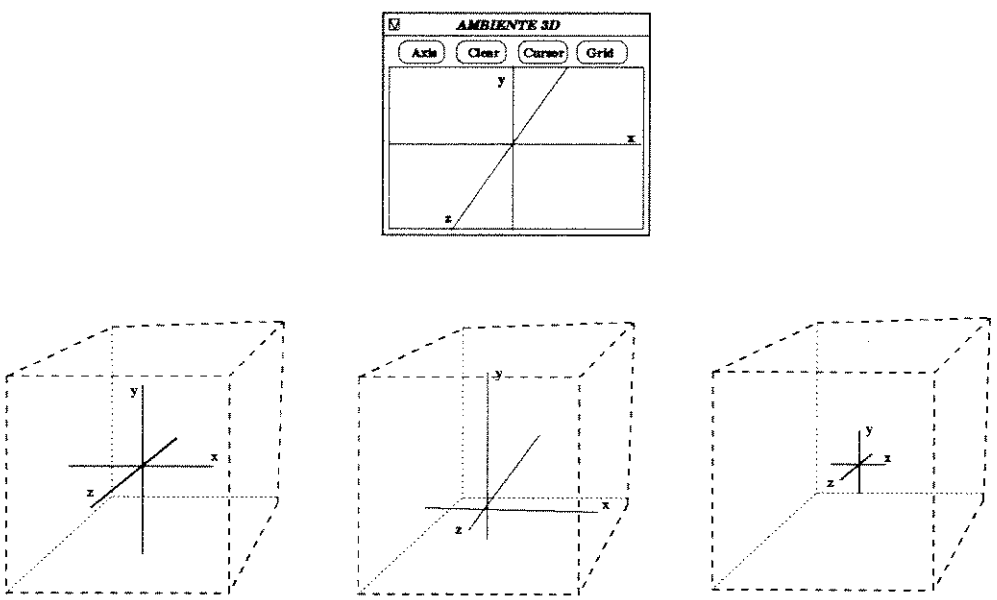

Figura A.4: *Cursor tridimensional*

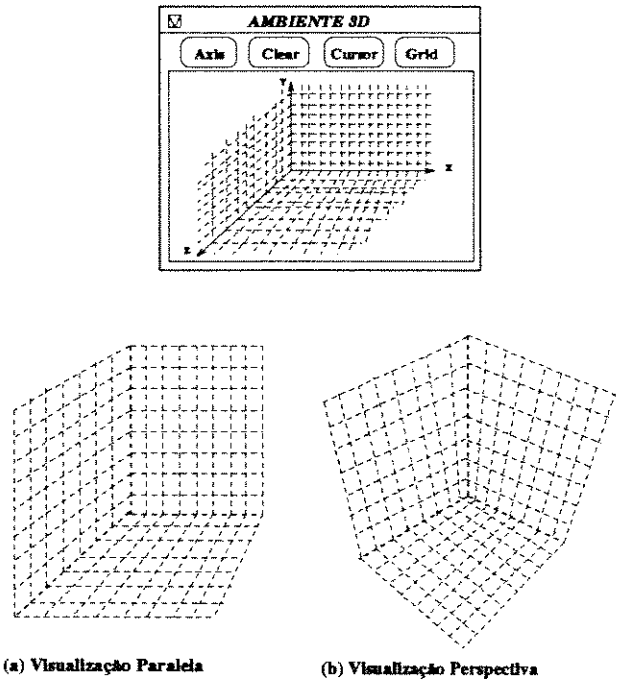

Figura A.5: *Grades tridimensionais*

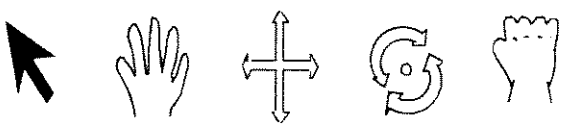

Figura A.6: *Exemplos de cursores 2D*

A.4 Feedback Textual

Número	Mensagem
1	Selecione um dos planos da grade 3D
2	Forneça os os três pontos que definem o *working plane*
3	Posicione objeto na cena
4	Objeto centrado na posicao [x, y, z]
5	Dimensione o objeto
6	Forneça os pontos de controle da curva

Tabela A.1: *Exemplos de mensagens para o feedback textual no ProSIm*

A.5 Tabelas de Variáveis e Funções

Nome	Significado
ProSImAppl	janela principal da aplicação *ProSIm*
ProSImIcon	ícone do *ProSIm*
ConfirmWindow	janela de confirmação utilizada em vários pontos do diálogo
Conf	confirmação do usuário (booleana)
ProSImMenu	menu principal do *ProSIm*
WkP	ferramenta de manipulação (*Working Plane*)
Grades3D	ferramenta visual (grades tridimensionais)
Eixos3D	ferramenta visual (eixos tridimensionais)
Cursor3D	ferramenta de manipulação (cursor tridimensional)
st	estado atual da interface
ObjBas	objeto básico
ObjCplx	objeto complexo formado pelas operações booleanas

Tabela A.2: *Exemplos de variáveis utilizadas na especificação do ProSIm*

Nome	Significado
exibe(x)	apresenta o objeto x numa posição específica do *workspace*, sobrepondo todas as janelas ativas que o interceptam
reexibe(x)	apaga o objeto x de uma posição, apresentando-o em nova posição
apaga(x)	apaga o objeto x da tela
selec(x)	seleciona o objeto x na tela. Esta seleção pode ser especificada em níveis mais baixos de abstração, por exemplo, indicando se a seleção vai ser feita num menu *toggle* ou através de manipulação direta. Normalmente, essa decisão vai estar relacionada na coluna de conexão com a computação.
EJanela(w)	função booleana que retorna verdadeiro se w é uma janela, e falso caso contrário
SalvaAmbiente(w)	salva o *status* global da aplicação rodando na janela w
MudaCursor2D (tipo)	utilizada no feedback da interface, muda a aparência do cursor dependendo da operação que for realizada (ver Figura A.6).

Tabela A.3: *Exemplos de funções utilizadas na especificação do ProSIm*

Apêndice B

Tcl/Tk

Tcl é uma linguagem para desenvolvimento de interfaces, de domínio público, desenvolvida por John Ousterhout, da Universidade de *Berkeley*, Califórnia. Consiste de uma linguagem interpretada, que pode ser combinada com programas C (i.e., um programa *Tcl* pode chamar programas C e um programa C pode definir e invocar funções *Tcl* - nesse caso, o interpretador *Tcl* é ligado como uma subrotina ao executável em C). *Tcl* funciona primariamente em *Unix*, mas o porte para outras plataformas está em andamento. *Tk* é uma *toolkit* escrita em *Tcl* para criar interfaces gráficas com o padrão parecido com o *Motif*[1]

A título de ilustração, apresenta-se a seguir alguns trechos de código escrito em *Tcl/Tk*, que implementam a ferramenta *QUANTUM*:

B.1 quantum.tcl

```
#-------------------------------------------------------------------------
# Filename : quantum.tcl
# Author   : Matt Jackson / Arcel Castillo / Brian Amento
# Modified by Elton Jose da Silva (05/22/95) Unicamp - Brazil
# Purpose  : main code file for QUANTUM prototype
# Copyright 1992, 1993 Virginia Polytechnic Institute and State University.
# All rights reserved.
#include <tk.h>
#-------------------------------------------------------------------------
```

[1] *Tk* é totalmente independente das bibliotecas escritas para o *Motif*, apesar de fornecer a mesma aparência e comportamento.

```
global auto_path bit_path ideal_lib
set tk_strictMotif 1
set ideal_lib "/home/faculty/beatriz/uan/ideal-beta"
set auto_path "/home/msc/elton/tese/quantum $ideal_lib/lib $auto_path"
set bit_path "$ideal_lib/Bitmaps"
init_ideal

#-------------------------------------------------------
# set all global variables here
#-------------------------------------------------------

### array holding UAN macros
global UANmacro
set UANmacro(1) "^\[X\]"
set UANmacro(2) "^3D\[X\]"
set UANmacro(3) "MLv^"
set UANmacro(4) "MLw"
set UANmacro(5) "MCv^"
set UANmacro(6) "MRv^"
set UANmacro(7) "selec(X)"
set UANmacro(8) "\@(x,y,z)"
set UANmacro(9) "selec(X)"
set UANmacro(10) "exibe(X)"
set UANmacro(11) "reexibe(X)"
set UANmacro(12) "apaga(X)"
set UANmacro(13) "apaga(X)"
set UANmacro(14) "apaga(X)"

### lists of task colors to use
global task_colors_l
set task_colors_l {tomato pink orange yellow YellowGreen ForestGreen}
global task_colors_r
set task_colors_r {SlateBlue LightSkyBlue peru orchid gray snow}

### coordinates for next window to display
global next_win_x
set next_win_x 200
global next_win_y
set next_win_y 200
```

```
### dimensions of Abstraction Hierarchy canvas
global wkspc_width
set wkspc_width 75
global wkspc_height
set wkspc_height 50


#-------------------------------------------------------
# deleteTask - delete task icon and all links
#-------------------------------------------------------
proc deleteTask {c} {
   ### find the task icon and delete
   set icon [getIconNum $c selected]
   $c delete $icon

   ### delete links from this icon
   set link_list [$c find withtag A$icon]
   foreach link $link_list {
      $c delete $link
   }

   ### delete links to this icon
   set link_list [$c find withtag B$icon]
   foreach link $link_list {
      $c delete $link
   }
}

############################################################
# main
############################################################

### hide root window
wm withdraw .

### we are busy
busyWindow on

### create help and reference windows
```

```
createAboutWin .aboutwin
busyWindow tick
createHelpWin .helpwin
busyWindow tick
createUanRef .uanref
busyWindow tick

### create palettes
doHierpal .hierpal
busyWindow tick
doNotepal .notepal
busyWindow tick
doUANpal .uanpal
busyWindow tick

### create menubar
createMenus .menupal .ah.workarea.c
busyWindow tick

### create Abstraction Hierarchy
createAbsHier .ah
busyWindow tick

### create Message Window
createMessageBox .msgbox
placeWin .msgbox 345 574
raise .msgbox
busyWindow tick

### show Abstraction Hierarchy
showAbsHier
tk_bindForTraversal .ah
focus .ah

### we are ready
busyWindow off

writeMessage "QUANTUM loaded successfully"
```

B.2 palettes.tcl

```
# ------------------------------------------------------------------------
# Filename : palettes.tcl
# Author   : Arcel Castillo / Matt Jackson / Brian Amento
# Modified by Elton Jose da Silva (05/15/95) Unicamp - Brazil
# Purpose  : code for creating and managing all palettes
# ------------------------------------------------------------------------

#-----------------------------------------------
# doUANpal - create UAN palette
#-----------------------------------------------
proc doUANpal {{w .uanpal}} {
   global UANpaltog
   global UANstring
   global UANmacro
   global new_string
   global bit_path
   global active_tool
   global text_font

   if {$UANpaltog == "OFF"} then {
      set UANpaltog "ON"
      if {[winfo exists $w] == 1} then {
         wm deiconify $w
      } else {
         ### set palette features
         set ht 1
         set wd 10
         set num_macros 14
         set pal_color LightGray

         ### make the palette
         toplevel $w -bg $pal_color -relief raised -bd 2
         wm geometry $w +847+100
         wm title $w "UAN"
         wm iconname $w "UAN"
         wm overrideredirect $w 1
         wm withdraw $w
         wm deiconify $w
         tkwait visibility $w
```

```
# ------------------------------------------------
# changeCursor  - change cursor shape to reflect current action
# ------------------------------------------------
proc changeCursor {c win_type} {
   global active_color
   global active_tool
   global bit_path

   case $win_type in {
      {ah} {
         case $active_tool in {
            {pointer} {
               set actcurr "@[set bit_path]/read_note.bmp black"
            }
            {place delete alias link rename \
            fanin copy cut locktog fanout} {
               set actcurr "@[set bit_path]/[set active_tool]_32.bmp black"
            }
            {text sketch audio video issue} {
               set actcurr "@[set bit_path]/place_[set active_tool]_note.bmp black"
            }
            {readnote} {
               set actcurr "@[set bit_path]/read_note.bmp black"
            }
            {default} {
               set actcurr "@[set bit_path]/invalid_32.bmp red"
            }
         }
      }
      {notebar} {
         case $active_tool in {
            {pointer} {
               set actcurr "@[set bit_path]/read_note.bmp black"
            }
            {text sketch audio video issue} {
               set actcurr "@[set bit_path]/place_[set active_tool]_note.bmp black"
            }
            {default} {
               set actcurr "@[set bit_path]/invalid_32.bmp red"
```

```
            }
         }
      }
      {td} {
         case $active_tool in {
            {pointer} {
               set actcurr "@[set bit_path]/read_note.bmp black"
            }
            {insertion} {
               set actcurr "@[set bit_path]/insert.bmp black"
            }
            {insertUANstr*} {
               set actcurr "@[set bit_path]/insertUANstr.bmp MediumBlue"
            }
            {text sketch audio video issue} {
               set actcurr "@[set bit_path]/place_[set active_tool]_note.bmp black"
            }
            {default} {
               set actcurr "@[set bit_path]/invalid_32.bmp red"
            }
         }
      }
      {lib} {
         case $active_tool in {
            {pointer} {
               set actcurr "@[set bit_path]/read_note.bmp black"
            }
            {insertion} {
               set actcurr "@[set bit_path]/insert.bmp black"
            }
            {default} {
               set actcurr "@[set bit_path]/invalid_32.bmp red"
            }
         }
      }
   }

   $c configure -cursor $actcurr
   update
}
```

B.3 taskDesc.tcl

```
# ------------------------------------------------------------------------
# Filename : taskDesc.tcl
# Author   : Matt Jackson / Arcel Castillo / Brian Ametno / Craig Struble
# Modified by Elton Jose da Silva (03/02/95) Unicamp - Brazil
# Purpose  : code for creation and management of task descriptions
# ------------------------------------------------------------------------

#-----------------------------------------------------
# createTDwin - make a task description window and call spreadsheet widget
#-----------------------------------------------------
proc createTDwin {{c .ah} task_num color name {load false} {fname none}} {
   global bit_path
   global text_font
   global active_tool
   global blank_color

   ### create task description window
   set w .tbltask$task_num
   toplevel $w
   placeWin $w
   wm title $w "Task Description"
   wm iconname $w "Task Desc"
   wm maxsize $w 1024 1024
   wm withdraw $w

   ### define spreadsheet labels
   set labels {{"User Actions" 4c} {"Interface Feedback" 6c}\
      {"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}

   .
   .
   .
```

Apêndice C

Um exemplo de arquivo .qtm

```
*notes
*descriptions
5 "Cameras" tomato
spreadsheet $w  -labels {{"User Actions" 4c} {"Interface Feedback" 6c}
{"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}
-hscroll 15c -vscroll 5c
-font "-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*"

set cell_text { {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } }
4 "ModelagemGeom3D" orange
spreadsheet $w  -labels {{"User Actions" 4c} {"Interface Feedback" 6c}
{"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}
-hscroll 15c -vscroll 5c
-font "-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*"

set cell_text { {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } }
6 "Rendering" yellow
spreadsheet $w  -labels {{"User Actions" 4c} {"Interface Feedback" 6c}
{"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}
-hscroll 15c -vscroll 5c
-font "-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*"
```

```
set cell_text { {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } }
7 "Animacao" orange
spreadsheet $w  -labels {{"User Actions" 4c} {"Interface Feedback" 6c}
{"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}
-hscroll 15c -vscroll 5c
-font "-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*"

set cell_text { {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } }
3 "IniciarProSIm" LightSkyBlue
spreadsheet $w  -labels {{"User Actions" 4c} {"Interface Feedback" 6c}
{"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}
-hscroll 15c -vscroll 5c
-font "-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*"

set cell_text { {"~[ProSImIcon];" "" "" "" "" }
{"MLw | KOpen v^" "ProSImIcon-! ? ProSImIcon!\npt Icon <> ProSImIcon ?
Icon-!\n@x',y' exibe(ProSImMain)\napaga(ProSImIcon)" "" "st=ultimo estado
em que a\naplicacao foi deixada" "\n\nx',y'=ultima posicao em que\na janela
foi desativada" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } }
2 "FinalizarProSIm" snow
spreadsheet $w  -labels {{"User Actions" 4c} {"Interface Feedback" 6c}
{"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}
-hscroll 15c -vscroll 5c
-font "-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*"

set cell_text { {"~[QuitItem in ProSImMenu];" "" "" "" "" }
{"MLv^" "@x',y' exibe(ConfWindow)\n(Conf==sim)?
{\n  apaga(ConfWindow)\n  apaga(ProSImWindow) }
:\n  apaga(ConfWindow)\n" "" "st=estado final" "\npt w,
ProSImWindow(w) ?\n{ SalvaAmbiente(w)\n   apaga(w) }" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } }
1 "UtilizarProSIm" tomato
spreadsheet $w  -labels {{"User Actions" 4c} {"Interface Feedback" 6c}
```

```
{"Exceptions" 4c} {"Interface State" 6c} {"Computation Connection" 6c}}
-hscroll 15c -vscroll 5c
-font "-*-helvetica-bold-r-normal-*-14-*-*-*-*-*-*-*"

set cell_text { {"I" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" }
{"" "" "" "" "" } {"" "" "" "" "" } {"" "" "" "" "" } }
*taskicons
icon5 task5 268 381 "Cameras" tomato 0 0 {}
icon4 task4 105 168 "ModelagemGeom3D" orange 0 0 {}
icon6 task6 410 259 "Rendering" yellow 0 0 {}
icon7 task7 578 358 "Animacao" orange 0 0 {}
icon3 task3 149 304 "IniciarProSIm" LightSkyBlue 0 0 {}
icon2 task2 622 218 "FinalizarProSIm" snow 0 0 {}
icon1 task1 359 39 "UtilizarProSIm" tomato 0 0 {}
*links
icon1 icon3
icon1 icon2
icon1 icon7
icon1 icon6
icon1 icon5
icon1 icon4
*end
```

Apêndice D

Glossário básico de HCI

A seguir, é apresentado um conjunto de definições para termos-chave comumente utilizados no campo de *HCI*. Muitas das definições estão baseadas em [OliveiraJr93]. Os termos estão dispostos em ordem alfabética, entretanto, alguns deles podem referenciar outros apresentados mais adiante.

callback procedures: consistem de chamadas a procedimentos definidos pelo programador e executados pela aplicação em resposta às ações tomadas pelo usuário na interface em tempo de execução.

CUA: *Common User Access*, padrão lançado em 1987 pela *IBM* no pacote denominado *SAA (System Application Architeture)* com o objetivo de facilitar a utilização de software pelos usuários. Define basicamente métodos de interação homem-computador, incluindo recursos para atender desde usuários iniciante até os mais experientes. O aplicativo mais popular que segue as recomendações *CUA* é o *Windows 3.0* da *Microsoft*.

ergonomia: estudo científico dos problemas relativos ao trabalho humano, e que devem ser levados em conta na projeção de máquinas, equipamentos e ambientes de trabalho.

estilos de interação: coleção de *objetos* da interface e *técnicas* de interação associadas que podem ser escolhidas pelo projetista da interface. Alguns exemplos são janelas, menus, formulários, caixas de diálogo, linguagem de comando, interfaces gráficas, *touchscreens*, reconhecimento de voz, realidade virtual etc.

gadget: são implementações simplificadas de alguns *widgets Motif*, minimizando a conversação entre a aplicação e o servidor de janelas, aumentando o desempenho da aplicação.

groupware: refere-se a sistemas de computação que suportam grupos de pessoas engajadas numa tarefa comum, fornecendo uma interface para este ambiente compartilhado.

GUI: *Graphic User Interface*, refere-se a sistemas que permitem a criação e manipulação de interfaces através de eventos de janelas, menus, ícones, caixas de diálogo, mouse e teclado. *Microsoft Windows* e *X Windows* são exemplos de *GUIs*. Geralmente possuem como base ambientes multiprogramados e permitem associar cada janela a um processo.

GUIDE: *Graphical User Interface Development Environment*, ferramenta desenvolvida e comercializada pela *Sun Microsystems*. Suporta a prototipagem rápida de interfaces no padrão *OpenLook*.

guidelines de projeto: são sugestões de como produzir boas interfaces. Geralmente estão relacionados com consistência e simplicidade, considerações sobre a memória humana, cognição, feedback da interface, etc. Inevitavelmente, conflitos ocorrem quando se tenta aplicar *guidelines*, devendo-se fazer uma análise de compromissos.

hipertexto: tipo de interface onde o usuário seleciona um objeto em destaque, podendo navegar entre diferentes tipos de informação. Se o objeto selecionável for texto ou gráfico, a interface é chamada de *hipermídia*.

interface multimodal: que fornece ao usuário mais de uma modalidade (simultaneamente ou não) de comunicação com o sistema e/ou usa mais de uma modalidade (simultaneamente ou não) para comunicar informações ao usuário.

janela: (*window*), é uma área geralmente retangular na 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. Neste caso, cada janela pode estar associada a um programa sendo executado concorrentemente [Lucena92].

LTM, STM: a memória humana está dividida em memória de curta duração (*Short Term Memory*) e memória de longa duração (*Long Term Memory*). Uma vez iniciado o processamento do cérebro, a informação é copiada para uma pequena memória de trabalho (*STM*), que tem capacidade para 7 ± 2 *chunks*[1] simultâneos. O tempo de duração de determinada informação na *STM* varia de pessoa para pessoa e geralmente está na faixa de 30 segundos a 2 minutos. Quando uma informação permanece muito tempo na *STM*, ela é transferida para a memória de longa duração (*LTM*), que retém a informação indefinidamente. Uma discussão mais detalhada sobre este assunto pode ser encontrado em [Thimbleby90].

look and feel: é um termo genérico utilizado para expressar apresentação (aparência) e comportamento (forma de operação) de uma interface (ou de objetos da interface). Exemplos desses objetos são botões de escolha, telas de texto, *sliders* (controles deslizantes), chaves, ícones, etc. A esses conjuntos de objetos denominam-se padrões de interface homem-computador ou padrões de *look and feel*. Apesar de serem denominados *padrões*, nenhum deles foi ainda efetivamente recomendado por algum órgão nacional ou internacional de normatização, tais como ABNT, *ISO* ou *ANSI*.

menu: é uma lista de itens na qual uma ou mais seleções podem ser feitas pelo usuário. É um dos mais populares estilos de interação, pois reduz o número de memorizações (o usuário seleciona os itens da lista diretamente), reduzindo o número de erros possíveis.

Motif: *padrão* de interface homem-computador de propriedade da *OSF*.

[1] unidade básica de informação a ser relembrada, podendo ser um nome de pessoa, uma palavra, uma frase, um número ou um conceito mais amplo.

OpenLook: *padrão* de interface homem-computador de propriedade da *Sun Microsystems*.

OSF: *Open Software Foundation*, fundada em maio de 1988 pelas empresas *IBM, DEC, HP, Apollo, Bull, Nixdorf* e *Siemens*, com o principal objetivo de avaliar e definir padrões que garantissem a estas empresas a filosofia de sistemas abertos. Resposta à iniciativa da *Sun Microsystems* em desenvolver seus equipamentos de forma que eles se comunicassem com qualquer outro sistema de computação.

realidade virtual: termo freqüentemente utilizado para se referir a qualquer simulação interativa onde geralmente são explorados vários sentidos do ser humano (visão, audição, tato etc). Ao invés de usar o mouse para manipular objetos na tela, o usuário projeta-se em um ambiente tridimensional, ou *mundo virtual*, e pode manipular objetos através do uso de equipamentos especiais. O objetivo principal da realidade virtual é projetar sistemas de computação que se comuniquem com humanos de maneira também humana, ao invés de forçar os usuários a se conformarem com as restrições de comunicação do computador.

sistema de janelas: *Window Systems*, sistemas que permitem a execução de múltiplas aplicações em janelas diferentes. Pode-se utilizar programas que mesmo sendo executados em outras máquinas, têm os seus resultados enviados via rede como se estivessem sendo executados na máquina local. Um tipo básico de arquitetura de sistemas de janelas é baseado no *esquema cliente/servidor*.

gerenciador de janelas: *Window Manager*, aplicação que o usuário interage para executar as funções necessárias ao gerenciamento do trabalho em um ambiente com múltiplas janelas (criação, movimentação, alteração do tamanho, redução de uma janela a um ícone, destruição de uma janela, etc). Estas operações, normalmente, são o resultado da fatoração das operações necessárias a todas as aplicações que manipulam janelas. Exemplos de gerenciadores de janelas são o *Openwindows* e o *mwm* (*Motif Window Manager*).

sistema reativo: é um sistema que 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. Interfaces homem-computador constituem um exemplo de sistemas reativos. Este tipo de sistema ainda requer métodos e abordagens especiais para desenvolvimento, devido ao seu comportamento complexo.

toolkit: biblioteca de rotinas, utilizadas pelos programadores, para implementar a interface. A utilização de *toolkits* livra o programador de detalhes de mais baixo nível, pois encapsulam o *look and feel* de técnicas de interação utilizadas em aplicações interativas. Os tipos de dados ou objetos básicos utilizados para este encapsulamento são chamados de *widgets*. O código baseado numa *toolkit* é aproximadamente um quinto do código escrito utilizando-se a biblioteca do sistema de janelas *XWindow*, também conhecida com *XLib*. Exemplos de *toolkits* são o *XVIEW*, que segue o padrão *OpenLook* e o *XmMotif*, que segue o padrão *OSF/Motif*.

touchscreen: tela sensível ao toque, que substitui algumas entradas via teclado ou mouse. É um dos dispositivos de entrada de dados mais duráveis e simples de usar, por isso bastante utilizados em em sistemas de acesso ao público, como no *Epcot Center*.

UIDS: *User Interface Development System* é um *UIMS* de última geração integrado a ferramentas *CASE* e que provêem auxílio a analistas e programadores nas decisões de projeto da interface.

UIMS: *User Interface Management System* conjunto integrado de ferramentas de software que auxiliam programadores e analistas em todo o processo de desenvolvimento de interfaces, incluindo especificação, projeto, construção de protótipos, implementação, execução, avaliação, modificação e manutenção da interface. A utilização de um *UIMS* permite a rápida criação e alteração da interface e a simulação do diálogo, reduzindo o tempo de desenvolvimento. Exemplos de *UIMSs* são *SERPENT*, desenvolvido pelo *SEI*, o *TeleUSE*, desenvolvido e comercializado pela empresa *TeleSoft* e o *XVT*, desenvolvido pela *XVT Software Inc.*

usabilidade: do termo em Inglês, *usability*, refere-se à facilidade de utilização de um sistema.

X Window System: sistema de janelas implementado em consórcio das principais empresas fabricantes de workstations no mundo, entre elas *Apollo, AT&T, DEC, IBM e HP*, e coordenado pelo *MIT*.

XLib: biblioteca de interface C que permite a utilização do protocolo *X*. Provê basicamente rotinas gráficas, de criação e manipulação de janelas e de tratamento de eventos gerados através de dispositivos de entrada (mouse e teclado). A *XLib* fornece uma interface que esconde dos programadores detalhes de codificação do protocolo *X*, manipulando automaticamente os *buffers* de requisição de mensagens e minimizando as chamadas ao sistema.

widget: é um termo genérico utilizado para designar elementos primitivos utilizados no projeto de interfaces h-c. São técnicas de interação fornecidos por uma *toolkit*. Exemplos de *widgets* são textos, gráficos, caixas de diálogo, rótulos, menus, botões de escolha, *scrollbars*, *sliders* etc. Utilizando uma *toolkit*, o programador constrói a interface da aplicação através de instâncias de classes de *widgets*.

Referências Bibliográficas

[Bass91] LEN BASS e JOELLE COUTAZ. *Developing software for the user interface.* Addison-Wesley, 1991.

[Boehm76] B. BOEHM. Software engineering. *IEEE Trans. on Computers*, v. C-25, n. 12, pp. 1226–1241, Dezembro 1976.

[Boehm88] B. BOEHM. A spiral model for software development and enhancement. *Computer*, v. 21, n. 5, pp. 61–72, Maio 1988.

[Buxton86] WILLIAM BUXTON. There's more to interaction than meets the eye: some issues in manual input. In D. NORMAN e S. DRAPER, EDS., *User Centered System Design*, pp. 318–337. LEA, 1986.

[Card83] STUART K. CARD, THOMAS P. MORAN e ALLEN NEWELL. *The psichology of human-computer interaction.* LEA, 1983.

[Carolis94] BERARDINA DE CAROLIS e FIORELLA DE ROSIS. Modelling adaptive interaction of OPADE by petri nets. *SIGCHI Bulletin*, v. 26, n. 2, 1994.

[CarterJr91] JAMES A. CARTER JR. Combining task analysis with software engineering in a methodology for designing interactive systems. In JOHN KARAT, ED., *Taking Software Design Seriously*, pp. 209–233. Academic Press, 1991.

[Casaday91] GEORGE CASADAY. Balance. In JOHN KARAT, ED., *Taking Software Design Seriously*, pp. 45–61. Academic Press, 1991.

[Coad90] P. COAD e E. YOURDON. *Object-oriented analysis.* Yourdon Press, 1990.

[Figueiredo91] ANTÔNIO GONÇALVES FIGUEIREDO FILHO. *Um processo de síntese de sistemas reativos.* Tese de Mestrado, IMECC, Universidade Estadual de Campinas, Dezembro 1991. Mestrado em Ciência da Computação.

[Firth91] CHRIS FIRTH e RICHARD C. THOMAS. The use of command language grammar in a design tool. *Int. J. Man-Machine Systems*, v. 34, pp. 479–496, 1991.

[Fitts54] P. M. FITTS. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, v. 47, pp. 381–391, 1954.

[Foley89] JAMES FOLEY ET AL. Defining interfaces at a high level of abstraction. *IEEE Software*, pp. 25–32, Janeiro 1989.

[Foley90] JAMES D. FOLEY ET AL. *Computer graphics: principles and practice.* Addison-Wesley, 2a ed., 1990.

[Gane79] C. GANE e T. SARSON. *Structured systems analysis.* Prentice-Hall, 1979.

[Gong94] RICHARD GONG e DAVID KIERAS. A validation of the GOMS model methodology in the development of a specialized, commercial, software application. In *CHI'94 CONFERENCE PROCEEDINGS: Human factors in computing systems*, pp. 351–357. ACM, 1994. Austin.

[Green86] MARK GREEN. A survey of three dialogue models. *ACM Trans. Graph.*, v. 5, n. 3, pp. 244–275, Julho 1986.

[Grudin93] JONATHAN GRUDIN. Interface: an evolving concept. *Communications of ACM*, v. 36, n. 4, pp. 110–119, Abril 1993.

[Gugerty93] LEO GUGERTY. The use of analytical models in human-computer-interface design. *Int. J. Man-Machine Studies*, pp. 625–660, 1993.

[Harel87] D. HAREL. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, v. 8, n. 3, pp. 231–274, Junho 1987.

[Hartson89] H. REX HARTSON e DEBORAH HIX. Human-computer interface development: concepts and systems for its management. *ACM Computing Surveys*, v. 21, n. 1, pp. 5–92, Março 1989.

[Hartson90] H. REX HARTSON, ANTONIO C. SIOCHI e DEBORAH HIX. The UAN: a user-oriented representation for direct manipulation interface designs. *ACM Trans. Inf. Syst.*, v. 8, n. 3, pp. 181–203, Julho 1990.

[Hartson92a] JEFFREY L. BRANDENBURG H. REX HARTSON e DEBORAH HIX. Different languages for different development activities: behavorial representation techniques for user interface design. In BRAD A. MYERS, ED., *Languages for Developing User Interfaces*, pp. 303–326. Jones and Bartlett, Boston, 1992.

[Hartson92b] H. REX HARTSON e P. GRAY. Temporal aspects of tasks in the User Action Notation. In *Human-Computer Interaction.* Elsevier, 1992.

[Haunold94] PETER HAUNOLD e WERNER KUHN. A keystroke level analysis of a graphics application: manual map digitizing. In *CHI'94 CONFERENCE PROCEEDINGS: Human factors in computing systems*, pp. 337–343. ACM, 1994. Austin.

[Hix90] DEBORAH HIX e GEORGE CASADAY. Reports of the working group on interface design decisions and representation. *ACM SIGCHI Bulletin*, v. 22, n. 2, pp. 34–41, Outubro 1990.

[Hix93] DEBORAH HIX e H. REX HARTSON. *Developing user interfaces: ensuring usability through product and process.* John Wiley and Sons, 1993.

[Hooper86] KRISTINA HOOPER. Architetural design: an analogy. In DONALD A. NORMAN e STEPHEN W. DRAPER, EDS., *User centered system design: new perspectives on human-computer interaction*, pp. 9–23. LEA, 1986.

[Irving94] PETER POLSON SHARON IRVING e J. E. IRVING. A GOMS analysis of the advanced automated cockpit. In *CHI'94 CONFERENCE PROCEEDINGS: Human factors in computing systems*, pp. 344–350. ACM, 1994. Austin.

[Jacob86a] ROBERT J. K. JACOB. Using formal specifications in the design of a human-computer interface. In N. GEHANI e A. D. MCGETTRIK, EDS., *Software Specification Techniques*, pp. 209–222. Addison-Wesley, 1986.

[Jacob86b] ROBERT J. K. JACOB. A specification language for direct-manipulation interfaces. *ACM Trans. Graph.*, v. 5, n. 4, pp. 283–317, Outubro 1986.

[James91] M. GREGORY JAMES. PRODUSER: PROcess for Developing USER interfaces. In JOHN KARAT, ED., *Taking design seriously: practical techniques for human-computer interaction design*, pp. 235–255. Academic Press, 1991.

[Karat90] JOHN KARAT e TOM DAYTON. Taking design seriously: exploring techniques useful in HCI design. *ACM SIGCHI Bulletin*, v. 22, n. 2, pp. 26–33, Outubro 1990.

[Karwowski92] WALDEMAR KARWOWSKI e GAVRIEL SALVENDY. Fuzzy-set-theoretic applications in modeling of man-machine interactions. In *Application of Fuzzy System Theory in Human Factors*, pp. 201–220. Academic Press, 1992.

[Khoshafian90] SETRAG KHOSHAFIAN. *Object orientation: concepts, languages, databases and user interfaces.* Addison-Wesley, 1990.

[Kieras88] DAVID E. KIERAS. Towards a pratical GOMS model methodology for user interface design. In M. HELANDER, ED., *Handbook of Human-Computer Interaction*, pp. 135–157. Elsevier, 1988.

[Kieras94] DAVID E. KIERAS. GOMS modeling of user interfaces using NGOMSL. In *CHI'94 CONFERENCE PROCEEDINGS: Human factors in computing systems*, pp. 371–372. ACM, 1994. Austin.

[Laurel86] BRENDA LAUREL. Interface as mimesis. In DONALD A. NORMAN e STEPHEN W. DRAPER, EDS., *User centered system design: new perspectives on human-computer interaction*, pp. 67–85. LEA, 1986.

[Laurel90] BRENDA LAUREL, ED. *The art of human-computer interface design.* Addison-Wesley, 1990.

[Lewis90] C. LEWIS ET AL. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In *CHI'90 CONFERENCE PROCEEDINGS: Human factors in computing systems*. ACM, 1990.

[Lim90] K. Y. LIM, J. B. LONG e N. SILCOCK. Integrating human factors with structured analysis and design methods: an enhanced conception of the Jackson System Development method. In D. DIAPER ET AL., EDS., *INTERACT'90*, pp. 225–230. Elsevier, 1990.

[Lucena92] FÁBIO N. LUCENA. *Construção de interfaces homem-computador: o uso de estadogramas na especificação e implementação de controle de interface*. Tese de Mestrado, IMECC, Universidade Estadual de Campinas, Março 1992. Mestrado em Ciência da Computação.

[Malheiros93] MARCELO DE GOMENSORO MALHEIROS. Uma interface gráfica para ProSIm. Relatório Técnico 92/4982-4, Universidade Estadual de Campinas, 1993. Relatório de Estágio de Iniciação científica submetido à FAPESP.

[Malheiros94] MARCELO DE GOMENSORO MALHEIROS. Uma interface gráfica para ProSIm. Relatório técnico, Universidade Estadual de Campinas, 1994. Relatório Final de Estágio de Iniciação científica submetido à FAPESP.

[Marcus91] A. MARCUS e A. DAM. User interface development for nineties. *IEEE Computer*, Setembro 1991.

[Mark86] WILLIAM MARK. Knowledge-based interface design. In D. NORMAN e S. DRAPER, EDS., *User Centered System Design*, pp. 219–242. LEA, 1986.

[Matias93] EDGAR MATIAS, I. SCOTT MACKENZIE e WILLIAM BUXTON. Half-QWERTY: a one-handed keyboard facilitating skill transfer from QWERTY. In *CHI'89 CONFERENCE PROCEEDINGS: Human factors in computing systems*, pp. 88–94. ACM, 1993.

[McDermid91] JOHN A. MCDERMID e PAUL ROCK. Software development process models. In JOHN A. MCDERMID, ED., *Software engineer's reference book*. Butterworth-Heinemann Ltda, 1991.

[Monk93] ANDREW MONK ET AL. *Improving your human-computer interface: a practical technique*. Prentice Hall, 1993.

[Moran81] T. MORAN. The command language grammar: a representation for the user interface of interactive computer systems. *Int. J. Man-Machine Systems*, v. 15, n. 1, pp. 3–51, Julho 1981.

[Myers89] BRAD A. MYERS. User-interface tools: introduction and survey. *IEEE Software*, v. 6, n. 1, pp. 15–23, Janeiro 1989.

[Norman86a] DONALD A. NORMAN e STEPHEN W. DRAPER, EDS. *User centered system design: new perspectives on human-computer interaction.* LEA, 1986.

[Norman86b] DONALD A. NORMAN. Cognitive engineering. In D. NORMAN e S. DRAPER, EDS., *User Centered System Design*, pp. 31–61. LEA, 1986.

[OliveiraJr93] NILTON DE OLIVEIRA JR. Glossário de termos em tecnologias de interface homem-máquina. Documento de circulação interna do CPqD/Telebrás, 1993.

[Payne86] STEPHEN J. PAYNE. Task-action grammars: a model of the mental representation of task languages. *Human Computer Interaction*, v. 2, pp. 93–133, 1986.

[Payne89] STEPHEN J. PAYNE. The structure of command languages: an experiment on task-action grammar. *Int. J. Man-Machine Studies*, v. 30, pp. 213–234, 1989.

[Pressman92] ROGER S. PRESSMAN. *Software engineering: a practitioner's approach.* McGraw-Hill, 3ª ed., 1992.

[Reinhardt94] ANDY REINHARDT. Building the data highway. *Byte*, v. 19, n. 3, pp. 46–74, Março 1994.

[Reisner81] PHYLLIS REISNER. Formal grammar and human factors design of an interactive graphics system. *IEEE Trans. Soft. Eng.*, v. SE-7, n. 2, pp. 229–240, Março 1981.

[Scheifler86] ROBERT W. SCHEIFLER e JIM GETTYS. The X window system. *ACM Transactions on Graphics*, v. 5, n. 2, Abril 1986.

[Shneiderman83] BEN SHNEIDERMAN. Direct manipulation: a step beyond programming languages. *IEEE Computer*, v. 16, n. 8, pp. 57–69, 1983.

[Shneiderman92] BEN SHNEIDERMAN. *Designing the user interface: strategies for effective human-computer interaction.* Addison-Wesley, 2ª ed., 1992.

[Silva93] ELTON JOSÉ DA SILVA e SIDNEY PIO DE CAMPOS. Desenvolvimento de uma interface gráfica para um sistema de modelagem e visualização de objetos. Relatório apresentado à disciplina IA-725 Computação Gráfica, da FEE/Unicamp, Agosto 1993.

[Siochi89] ANTONIO C. SIOCHI e H. REX HARTSON. Task-oriented representation of asynchronous user interfaces. In *CHI'89 CONFERENCE PROCEEDINGS: Human factors in computing systems*, pp. 183–188. ACM, 1989. Austin.

[Siochi91] ANTONIO C. SIOCHI, DEBORAH HIX e H. REX HARTSON. The UAN: a notation to support user-centered design of direct manipulation interfaces. In JOHN KARAT, ED., *Taking Software Design Seriously*, pp. 157–194. Academic Press, 1991.

[Thimbleby90] HAROLD THIMBLEBY. *User interface design.* ACM Press, 1990.

[Velasquez95] DÉLIA PERLA VELASQUEZ ALEGRE. *Técnicas de interação 3D através do mouse.* Tese de Mestrado, FEE, Universidade Estadual de Campinas, 1995. Mestrado em Engenharia Elétrica (em andamento).

[Venolia93] DAN VENOLIA. Facile 3D direct manipulation. In *CHI'93 CONFERENCE PROCEEDINGS: Human factors in computing systems*, pp. 348–355, Amsterdam, April 1993. ACM.

[Vilela94a] PLÍNIO ROBERTO SOUZA VILELA. *Uma ferramenta para auxílio visual ao teste e depuração de programas.* Tese de Mestrado, FEE, Universidade Estadual de Campinas, Março 1994. Mestrado em Engenharia Elétrica.

[Vilela94b] PLÍNIO ROBERTO SOUZA VILELA ET AL. Visualização de grafos de programa: uma abordagem sem reposicionamento. In *SIBGRAPI'94*, pp. 189–196, 1994.

[Wellner93] PIERRE WELLNER. Interacting with paper on the digitaldesk. *Communications of ACM - Computer Augmented Environments: Back to the real world*, v. 36, n. 7, pp. 87–96, Julho 1993.

[Wilson88] JAMES WILSON e DANIEL ROSENBERG. Rapid prototyping for user interface design. In M. HELANDER, ED., *Handbook of Human-Computer Interaction*, pp. 859–875. Elsevier, 1988.

[Yourdon79] E. YOURDON e L. CONSTANTINE. *Structured design.* Prentice-Hall, 1979.