Análise e Projeto de Ambientes Interativos

Lucas Martins do Amaral
Roberto da Silva Bigonha

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE CIÊNCIAS EXATAS
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Abstract

Um sistema de computação perde totalmente seu valor quando o homem não pode tirar proveito da capacidade computacional que lhe é fornecida. Muitas vezes o sucesso de um *software* é determinado pela facilidade ou dificuldade com que as funções apresentadas são alcançadas. Neste processo de comunicação entre duas entidades distintas, homem e máquina, está claro que o computador é quem deve se adaptar a forma de comunicação do homem. Este artigo investiga aspectos da interação homem-máquina e influências desta interação no desenvolvimento de ambientes interativos [Ama92].

[**keywords**]: interface homem-máquina, *toolkits*, UIDS, erros, ambiente interativo, fatores humanos, usuário.

1 Introdução

A construção de um sistema de *software* é uma tarefa complexa, mesmo quando o sistema é puramente computacional¹ sem interação com o mundo externo. A inserção do homem neste sistema representa um novo fator a ser considerado. Fator este, dinâmico, exigindo-se *interfaces* homem-máquina² de qualidade, o que torna ainda mais complicada a tarefa de desenvolvimento de um sistema de software. Um sistema interativo—aquele em que há interação entre homem e máquina—é avaliado não somente por sua capacidade computacional, mas também por sua capacidade de comunicação. Esta característica é fundamental para que os usuários possam interagir eficientemente com o sistema a fim de tirar proveito da capacidade computacional.

É frustrante saber que ao longo do processo de construção de um sistema interativo gasta-se mais tempo na produção da interface homem-máquina do que na programação da parte computacional da aplicação. Justifica-se, portanto, a necessidade de se estudar técnicas para projeto de interfaces homem-máquina como uma área em separado dentro do processo de desenvolvimento de software.

¹do inglês: *computer bounded*.

²do inglês: *Human-computer Interface—HCI*

Quando o conceito de interface começou a surgir, associava-se a interface ao *hardware* e software através dos quais homem e computador podiam se comunicar. Com a evolução do conceito, incluiu-se aspectos emocionais e cognitivos³ da experiência do usuário.

Para muitos, interface entre homem e computador é representada por terminais *teletype—tty*. Outros já associam a palavra interface a sistemas de janelas, onde é possível mapear, desenhar e manipular objetos.

Consideraremos como sendo interface, não só o meio físico que separa duas entidades distintas—homem e computador—em seu processo de comunicação. Deve-se encará-la como um meio intermediador 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 [Lau90], e portanto conhecimento sobre a maneira como as pessoas resolvem problemas.

O desenvolvimento de interfaces homem-máquina de qualidade se tornou possível a partir do surgimento de hardware capaz de sustentar as necessidades de interação entre homem e máquina. A medida que novas tecnologias surgem ou se tornam economicamente viáveis, interfaces mais poderosas são desenvolvidas utilizando recursos como som e imagem animada.

2 Desenvolvimento de HCI

O processo de desenvolvimento da interface homem-máquina de qualquer aplicação não difere do processo de desenvolvimento de outro software. Dois importantes aspectos podem ser observados:

1. O projeto e desenvolvimento da interface com o usuário é um processo iterativo de refinamento que dura enquanto existir motivação para modificações;
2. Em cada lado da interface existem representações abstrata dos componentes. Do lado do homem, essas abstrações são estudadas por psicólogos. Do lado da máquina, essas abstrações são descritas através de uma hierarquia de máquinas abstratas⁴ que fornecem vários níveis de serviço possuindo custos diferentes para implementação e manutenção.

Em geral, existem cinco papéis a serem desempenhados durante o desenvolvimento de um sistema interativo: usuário ou operador; projetista de sistemas; projetista da HCI; projetista da aplicação e programador do software da aplicação (veja [BC91]). Uma mesma pessoa pode muitas vezes desempenhar todos esses papéis.

1. *usuário ou operador* é aquele que utiliza o sistema interativo final.
2. *projetista do sistema* define a arquitetura global, especificando quais tarefas serão realizadas por quais partes do sistema.

³aspectos cognitivos se referem à aquisição de conhecimento

⁴veja Seção 3.

3. *projetista da HCI* é aquele que utiliza a especificação das tarefas feitas pelo projetista do sistema para definir a interface com a qual o operador irá interagir.
4. *projetista da aplicação* cria a estrutura de software necessária para implementar as tarefas realizadas pelo sistema, excluindo a interação homem-máquina.
5. *programador do software da interface* estrutura o software que irá implementar a interface com o usuário definida pelo projetista da HCI.

3 Um Modelo de Camadas

A interface representa para o usuário uma máquina abstrata com a qual ele irá interagir [BC91]. Uma forma de estruturar o software é através de uma hierarquia de máquinas abstratas, cada qual fornecendo serviços para camadas acima no estilo *cliente/servidor* [SG86]. A divisão de software em várias camadas representa um mecanismo de se proteger cada camada de modificações nas camadas abaixo. Os princípios fundamentais da modularização e ocultação de informações formam a base da noção de máquinas abstratas. Uma possível divisão em camadas de um sistema interativo apresenta os seguintes níveis: acionador de dispositivos, gerenciador de recursos, objeto de interação, controlador de diálogo e núcleo funcional. Esses níveis são descritos da seguinte forma:

1. *acionador de dispositivos* controla os dispositivos físicos. Exemplos de tais acionadores são aqueles que enviam instruções para exibir *pixels* na tela;
2. *gerenciador de recursos* controla os recursos reais envolvidos na interação. Ele não só gerencia os recursos físicos, mas também fornece abstrações usadas durante a interação. Sistemas de janelas são gerenciadores de recursos;
3. *objeto de interação* é uma entidade que o operador pode perceber e manipular com dispositivos físicos de interação como um *mouse* ou o teclado. Menus são exemplos de objetos de interação;
4. *controlador de diálogo* é a parte do software da HCI que controla a *media* e o seqüenciamento das interações do usuário. Este controlador determina a lógica das interações;
5. *núcleo funcional* implementa o conhecimento do problema. Este nível deve, idealmente, ser totalmente ignorante sobre a forma como suas estruturas de dados são apresentadas ao usuário. Não se deve, entretanto, esquecer que uma pessoa estará usando o sistema, e portanto, que erros poderão acontecer.

Em geral, o código da interface de uma aplicação pode representar uma fração significativa do código total de um sistema, chegando a 50% [BMS86]. Para acelerar o processo de desenvolvimento da HCI de um sistema interativo e conseqüentemente evitar o tratamento de alguns dos níveis descritos na hierarquia acima, utiliza-se ferramentas para projeto e desenvolvimento de interfaces homem-máquina.

4 Ferramentas para HCI

A utilização de ferramentas para HCI (*User-Interface Tools*) apresenta várias vantagens sob dois aspectos principalmente:

1. Obtêm-se melhores interfaces:
 - Projetos podem ser rapidamente prototipados e implementados, mesmo antes de se escrever o código da aplicação.
 - É mais fácil incorporar modificações após testes feitos pelo usuário.
 - Uma aplicação pode ter várias interfaces.
 - Aplicações diferentes apresentarão interfaces consistentes.
 - Torna-se fácil analisar estilos diferentes para uma interface
2. É mais fácil criar e mais econômico manter o código da interface.
 - O código será melhor estruturado e mais modular, pois ele foi separado da aplicação.
 - O código será mais reutilizado, pois partes comuns são incorporadas.
 - A confiabilidade da interface é maior.
 - Especificações da interface podem ser representadas, avaliadas e validadas facilmente.
 - Código específico de dispositivos está isolado na ferramenta.

Ferramentas para HCI dividem-se em duas formas gerais: *user-interface toolkits* e *user-interface development systems* (UIDS), ferramentas e sistemas de desenvolvimento, respectivamente.

Toolkits são bibliotecas de técnicas de interação, como menus, *scroll-bars*, botões, etc. Um programador usa um *toolkit* escrevendo código para chamar e organizar as técnicas de interação. *Toolkits* não facilitam o projeto da interface, nem a especificação do seqüenciamento e controle do diálogo. No entanto, a maior parte das ferramentas existentes são *toolkits*.

Um sistema de desenvolvimento de interfaces (UIDS) é um conjunto integrado de ferramentas projetadas para auxiliar o rápido desenvolvimento, ajuste e gerenciamento de interfaces para as mais diversas aplicações. Um UIDS é muito parecido com linguagens de quarta geração, onde se concentra na especificação do que fazer em vez da codificação de como fazer. Entre as ferramentas que integram um UIDS encontra-se um *toolkit*.

4.1 HCI Toolkits

Existem dois tipos de *toolkits*. O mais convencional é uma coleção de rotinas que podem ser chamadas pelos programas aplicação. O outro tipo usa o estilo de programação orientada a objetos com herança, o que torna bastante fácil para o projetista personalizar as técnicas de interação. Entre os *toolkits* existentes, o mais utilizado é o do sistema

X Window, Xt, que apesar de fornecido para linguagem C foi desenvolvido segundo o paradigma da programação orientada a objetos.

As desvantagens em se usar *toolkits* são o fato de eles proverem limitados estilos de interação, sua criação ser bastante cara e a utilização difícil.

4.2 Sistemas de Desenvolvimento de HCI

As deficiências dos *toolkits* levaram ao desenvolvimento dos UIDSs. As funções de um sistema de desenvolvimento são: a) controlar o *mouse* e outros dispositivos de entrada; b) validar entrada do usuário; c) tratar erros do usuário; d) processar *aborts* e *undos*; e) fornecer resposta apropriada mostrando que a entrada foi recebida; g) fornecer ajuda⁵ e *prompts*; h) atualizar a tela quando dados da aplicação forem alterados; i) informar a aplicação quando o usuário atualizar dados a ela referentes; j) cuidar da editoração de janelas; i) isolar a aplicação das rotinas de gerenciamento da tela e avaliar automaticamente a interface e propondo melhorias.

Para fornecer as funções descritas, um sistema de desenvolvimento deve conter: a) um *toolkit*; b) um componente de controle do diálogo; c) uma estrutura de programação; d) um editor de *layout* através de *mouse* para determinar a localização dos elementos da interface e e) um componente de avaliação da interface.

4.3 Deficiências das Ferramentas

Embora a utilização de *toolkits* e UIDSs esteja aumentando, vários problemas a tem limitado como dificuldade de utilização, dificuldade de especificação, pouca funcionalidade oferecida, dependência de linguagens ou sistemas, etc:

- Dificuldade de utilização. *Toolkits* tipicamente contêm centenas de rotinas que interagem sutilmente. A maioria dos UIDSs exigem que o projetista aprenda uma nova linguagem de uso específico. UIDSs gráficos ou de geração automática são em sua maioria experimentais.
- As especificações de interfaces são difíceis de compreender e editar. As linguagens usadas em muitos UIDSs para especificar a interface são pouco estruturadas (utilizam excessivamente variáveis globais, fluxo de controle não local, e *gotos* explícitos).
- Oferecem pouca funcionalidade. A maioria trata apenas de uma pequena parte da tarefa de projeto. Enquanto são muito bons lidando com menus, *scroll bars*, eles raramente podem ser usados para ajudar no controle da exibição e manipulação dos dados da aplicação.
- Em sua maioria não são transportáveis, o que implica na necessidade de refazer todo o trabalho de desenvolvimento da interface caso a aplicação migre para um ambiente operacional diferente.

⁵do inglês: *help*

- Não propiciam avaliação automática de projetos. As regras de estilos de interface existentes (Open Look [SA90] e Motif [Fou89]) estão apenas no papel, ou seja, o projetista da interface é quem está encarregado de desenvolver o projeto de acordo com as regras. As ferramentas a serem utilizadas durante a programação da interface não garantem que o produto final estará dentro das regras definidas pelo estilo adotado.
- Se por um lado agradam a projetistas que querem se livrar dos detalhes de estilo, e permitem coerência entre interfaces de várias aplicações, desagradam aqueles que desejam dar um estilo único aos seus produtos. Há perda de flexibilidade em troca da coerência visual.
- Frequentemente degradam o desempenho da aplicação.
- É bastante difícil separar a interface da aplicação, sendo muitas vezes necessário adotar soluções *ad-hoc* quando o UIDS limita a ação de rotinas da aplicação sobre as da interface.

4.4 Características das Ferramentas

As interfaces geradas utilizando-se qualquer ferramenta devem possuir as seguintes características [B⁺87]: consistência, adaptabilidade e imunidade à erros.

- *consistência* entre interfaces não significa que interfaces devam ser idênticas, mas sim que permitam ao usuário transferir conhecimento quanto a utilização de sistemas interativos de uma aplicação para outra;
- *adaptabilidade* se refere a possibilidade da interface ser alterada no que diz respeito a forma e estilo de diálogo para se adequar a vários tipos de usuários, desde novatos a especialistas;
- *imunidade a erros*⁶ estabelece como a interface tratará erros e fará a recuperação dos mesmos a fim de garantir a continuidade da operação da aplicação.

5 Estilos de Interação

O tipo de controle e comunicação em um sistema de software interativo é bastante influenciado pelo estilo de interação a ser usado. Existem pelo menos duas metáforas [Har89] que descrevem os principais estilos de diálogo: mundo conversacional e mundo em modelo.

No típico mundo conversacional, o usuário descreve o que deseja fazer através de uma linguagem de comando. Este tipo de diálogo é normalmente descrito linguisticamente, com gramáticas e máquinas de estado finito.

⁶na Seção 10 será mostrada a importância dos erros para o desenvolvimento de interfaces de qualidade.

No mundo em modelo, o usuário mostra o que fazer manipulando representações visuais de objetos. Este é o estilo de *manipulação direta*. Este tipo de diálogo é normalmente descrito em termos de objetos e ações. Ele é geralmente mais fácil de ser usado, embora ambos sejam amplamente adotados.

Manipulação direta na metáfora de mundo em modelo é freqüentemente associada a diálogo *multi-thread*, significando a disponibilidade de várias tarefas em determinado instante do diálogo.

6 Modelo do Usuário

Um sistema interativo não existe sem o usuário, ou seja, se ninguém o utilizar ele terá falhado na sua principal função: interagir com o homem. Não se pode construir um sistema que seja fácil e agradável de usar sem identificar as características do usuário. Psicólogos insistem neste ponto: “o projeto de sistemas interativos deve ser centrado no usuário” [BC91]. A razão em se modelar o usuário se deve à necessidade do projetista decidir qual nível de auxílio sintático e semântico terá que ser fornecido.

7 Conhecimento Sintático e Semântico

Existem várias teorias de como o conhecimento é adquirido. Uma delas é a teoria que estabelece o modelo semântico/sintático do comportamento cognitivo do usuário [Sch87].

Conhecimento Sintático representa as convenções linguísticas que o usuário deve dominar para comunicar com o sistema interativo, tanto para dar ordens (entrada de expressões) quanto para interpretar respostas (saída de expressões). Assim, o conhecimento sintático é específico a cada sistema.

Conhecimento Semântico insere uma hierarquia de conceitos ou símbolos sobre fatos e procedimentos. Num sistema interativo, conhecimento semântico dependente do *domínio* é inerente às tarefas que a aplicação desempenha e conhecimento semântico dependente do sistema representa o conhecimento sobre o funcionamento do sistema interativo.

Os conceitos de conhecimento semântico e sintático estão relacionados à estrutura de camadas citada na Seção 3. As camadas relativas à HCI fornecem processamento sintático, enquanto as rotinas da aplicação estão a cargo do processamento semântico.

8 Classificação Geral dos Usuários

Quão fidedigna é a caracterização do usuário depende da capacidade de identificá-lo. Entre as técnicas usadas para identificar usuários [Sch87] estão entrevistas, questionários, observações de campo, gravação em vídeo da utilização da HCI pelo usuário,

e seções onde o usuário relata os motivos pelos quais ele está tomando certa atitude quando utiliza um sistema interativo.

Schneiderman, em [Sch87], organiza a comunidade de usuários em novatos, de conhecimento intermitente e frequentes. Esta classificação é baseada no conhecimento semântico e sintático do usuário quanto a aplicação e o software.

- *usuários novatos* não possuem qualquer conhecimento sintático, possivelmente pouco conhecimento semântico sobre softwares e conhecimento semântico superficial sobre o domínio da aplicação.
- *usuários de conhecimento intermitente* são aqueles que possuem bom conhecimento semântico sobre o domínio da aplicação e softwares, mas apresentam problemas quanto a recordar os aspectos sintáticos.
- *usuários frequentes* estão familiarizados com a sintaxe e a semântica do sistema.

Projetistas de softwares devem sempre ter em mente que o modelo mental dos usuários evolui com o tempo, de um extremo ao outro.

É comum pensar que um produto é adequado para certo público quando testes de laboratório indicam aceitação por este público. A validação de testes de laboratório deve ser feita com cuidado, pois conclusões errôneas podem comprometer o sucesso de um software.

9 Validação de HCI e Fatores Humanos

O objetivo da área de fatores humanos⁷ é adequar interfaces à forma como usuários reais no mundo real a utilizam, garantindo que a HCI, independentemente de estilo, sirva ao ser humano enquanto trabalha sob as regras do computador [TK89]. No entanto, quando se considera as limitações humanas na interação com máquinas, a necessidade de se compreender estas limitações para se alcançar um projeto de sucesso sempre é relegada a segundo plano. Até então, a estratégia de se desconsiderar as limitações humanas em projetos de software não tem comprometido o sucesso do mesmo, pois as pessoas são adaptáveis e o projeto de sistemas onde se desconsidera aspectos humanos tem sido a norma, havendo pouca pressão competitiva para construção de softwares que considerem as pessoas como elas realmente são.

9.1 Hiato Ecológico

Enquanto regras importantes para construção de software têm seguido princípios gerais da cognição humana, é em geral aceito que esses princípios são insuficientes para realmente prever como as pessoas irão reagir a determinado produto. Desta forma, testes empíricos representam uma importante etapa na tentativa de se construir softwares que serão utilizáveis.

⁷nos EUA esta área é conhecida como *human factors*, enquanto na Europa é denominada ergonomia. Ela estuda como o ser humano utiliza seus sentidos e membros para trabalhar com ferramentas.

Pesquisadores devem se preocupar não somente com a existência de relações putativas entre variáveis independentes e dependentes (ex. característica de uma interface e tempo para aprender a utilizá-la), mas também com a significância desta relação no contexto de uso real (ex. se o usuário irá se incomodar em gastar algum tempo para aprender a utilizá-la). Classicamente, existe o problema na psicologia experimental de validação externa de testes, ou seja, a possibilidade de ao se projetar um experimento (teste de laboratório) validar a generalização dos resultados para a população ou situação de interesse. Em psicologia cognitiva, a generalização de situações artificiais de laboratório para situações reais de interesse é denominada *validade ecológica*. Assim, Thomas e Kellogg, em [TK89], denominam a diferença entre os testes de laboratório e os do mundo real como sendo devido ao *hiato ecológico*.

Hiatos ecológicos podem ser criados de duas formas: eliminando-se influências importantes do contexto de utilização real e criando influências no contexto de utilização no laboratório. Essas duas formas não são mutuamente exclusivas.

A existência de hiatos ecológicos não significa que testes de laboratório nunca sejam apropriados ou úteis. Significa, no entanto, que esses testes devem ser projetados inserindo aspectos de validação ecológica e que outras formas de avaliação são necessárias para cobrir aspectos falhos nos testes de laboratório. Para avaliar a capacidade de um software ser utilizado no mundo real, resultados de laboratório devem ser generalizados segundo quatro áreas: usuários, tarefas, contexto do sistema e contexto de trabalho do usuário.

Hiato Devido ao Usuário: Hiatos ecológicos relativos ao usuário são causados pelas características do usuário—suas motivações, habilidades cognitivas, preferências e hábitos—que podem variar do laboratório para o ambiente de trabalho.

Hiato Devido à Tarefa: Nesta área está a diferença entre as tarefas que podem ser observadas nos laboratórios e as tarefas que os usuários realmente vão realizar no mundo real. Uma das causas deste problema é devido a má formulação dos testes, também conhecida como *falência do exame escolar* [TK89]. Na escola, estudam-se várias matérias como álgebra, trigonometria, história e biologia. Em cada matéria existem vários tópicos, como equações do segundo grau, fatoração, etc. De modo geral, quando se faz um teste após estudar equações do segundo grau, serão apresentados problemas que requerem apenas este conhecimento para serem resolvidos. A falha está no fato de na vida real os problemas não ocorrerem desta forma determinística. Não basta as pessoas saberem como usar uma ferramenta para resolver um problema, elas devem saber reconhecer quando um problema pode ser resolvido por uma ferramenta e quando não pode.

Hiato Devido ao Contexto do Sistema: Tipicamente, softwares são testados isoladamente no laboratório, embora no mundo real eles existirão em grandes ambientes computacionais coexistindo com outros produtos e vários ambientes operacionais. Uma das principais questões quanto a utilização de softwares se deve à consistência (ou falta

de consistência) entre várias versões de um produto rodando em ambientes diferentes ou mesmo entre diferentes produtos, que realizam as mesmas tarefas, num mesmo ambiente computacional.

Outro aspecto a ser considerado é o tempo de utilização do produto. No laboratório os usuários utilizarão o produto por algumas horas, enquanto na vida real o usarão por alguns anos. Alguns fatores considerados desvantajosos durante uma curta utilização de um software (ex. personalização de uma interface) podem ser bastante significantes para se determinar a aceitação e satisfação dos usuários a longo prazo.

Hiato Devido ao Contexto de Trabalho: Recomendações feitas sob aspectos de fatores humanos podem ser impraticáveis devido a impossibilidades no ambiente de trabalho. Uma diferença bastante acentuada entre testes de laboratório e ambiente de trabalho se deve à variedade de tarefas e situações em que o usuário estará envolvido a cada instante. As pessoas são bastante influenciadas pela situação social em que se encontram (ex. cargo ocupado). Em testes de laboratório o desempenho dos usuários não é influenciado por este fator. Aspectos culturais também devem ser considerados quando um produto se destina ao mercado internacional.

Independentemente da qualidade da interface homem-máquina de um software, sempre haverá possibilidade para que erros sejam cometidos pelo usuário. A seguir é apresentada uma classificação de erros, podendo estes ser causados tanto pelo usuário quando pela interface da aplicação. Uma vez compreendidos os motivos causadores de erros durante a interação entre homem e máquina, projetistas de sistemas interativos podem evitar mecanismos de interação considerados mais susceptíveis à erros.

10 Erros

Através da análise de classes de erros que as pessoas cometem quando estão usando um sistema de software, é possível desenvolver princípios de projeto que minimizem a ocorrência de erros e seus efeitos.

Norman, em [Nor83], denomina *intenção* o nível mais alto de especificação de uma ação que se deseja realizar. A intenção pode ser resultado de um processo consciente ou inconsciente de decisão. Um erro de intenção é chamado *equivoco*⁸, enquanto erro durante a realização da intenção é denominado *falha*⁹.

Falhas podem ser classificadas em pequenos conjuntos de classes, baseando-se nos mecanismos que mais provavelmente as causam. A classificação básica está fundamentada em um modelo simplificado do homem, onde se assume que qualquer intenção torna um número de esquemas ativos, cada qual com um valor de ativação e um conjunto de condições capazes de dispará-los [Nor83].

⁸do inglês: *mistake*

⁹do inglês: *slip*

10.1 Erros de Modos de Operação

Modos de operação estarão presentes em sistemas de software enquanto se desejar realizar um número de operações maior que o número de teclas especiais disponíveis. Uma grande classe de erros é a dos *erros de modo*, onde o usuário realiza uma operação correta num modo de funcionamento errado. Este tipo de erro é resultado da resposta e indicação inadequadas do estado do sistema.

Para identificar o estado do sistema ou modo, alguns sistemas usam linhas em destaque. Embora bem intencionada, tal indicação leva a outra classe de erros: *erros de representação*. O número de modos em um sistema pode ser minimizado através da utilização de menus e dispositivos de seleção como *mouse*.

10.2 Erros de Representação

Um erro de representação ocorre quando a especificação de uma ação é insuficiente e a ambiguidade resultante induz a um ato equivocado. Em geral esses erros ocorrem quando ações diferentes possuem descrições similares, tanto na especificação das ações quanto no tipo dos argumentos.

10.3 Erros de Inconsistência

Uma classe de erros ocorre quando alguém tenta deduzir a seqüência de uma ação e a faz imprópriamente, formando uma seqüência apropriada para uma outra ação. Isto ocorre principalmente devido a falta de consistência na estrutura dos comandos do sistema. A estrutura apropriada para um comando é completamente diferente para outro embora os dois tenham um formato parecido e realizam tarefas com objetivos semelhantes. Tirar conclusões a partir da relação de um sistema para outro é um método comum e poderoso do raciocínio humano, mas pode levar a erros devido ao mapeamento de um domínio para o outro não ser consistente.

10.4 Erros de Captura

Erros de captura ocorrem quando há superposição nas seqüências necessárias para realizar duas ações diferentes, especialmente quando uma é realizada muito mais freqüentemente que a outra.

10.5 Erro de Ativação

Erros de ativação são de duas classes: ações inapropriadas são realizadas ou ações apropriadas deixam de ser realizadas. A primeira ocorre quando uma seqüência inapropriada é ativada por estar relacionada a seqüência desejada. A não execução de uma ação normalmente ocorre devido a falha de memória, normalmente gerada por interferência de outros eventos entre o instante de preparação da intenção e da realização da ação.

10.6 Sistemas e Pessoas

As pessoas cometerão erros, mesmo nos sistemas melhor projetados, com o melhor treinamento e motivação. A principal maneira de se tentar minimizar erros está na minimização dos efeitos dos erros. Isto significa que ações devem ser reversíveis tanto quanto possível. Não é suficiente pedir ao usuário que ele confirme o desejo de realizar uma determinada tarefa, pois se este procedimento se torna rotineiro, a confirmação passa a ser feita automaticamente como se fizesse parte da seqüência de comandos. Algumas vezes pode-se simular a real execução de um comando, quando na verdade ela foi apenas adiada dando possibilidade de recuperação ao usuário (ex. utilização da lata de lixo para remoção de arquivos).

11 Conclusões

O desenvolvimento de software para interface homem-máquina de sistemas difere do desenvolvimento do software que implementa a funcionalidade de uma aplicação por este estar diretamente relacionado à pessoas. Assim, é necessário pensar não só em algoritmos para resolução de problemas, em definição de estruturas de dados ou em modularização de código, mas principalmente em quem vai utilizar o sistema e em como a funcionalidade deste será apresentada. O usuário é o responsável pelo sucesso e pelo fracasso de um software, pois se ele não utilizá-lo, seu desenvolvimento terá sido em vão. Logo, quando se desenvolve programas interativos, deve-se ter em mente que as pessoas são diferentes, que componentes culturais e sociais afetarão o posicionamento do usuário frente a um produto. Desta forma, interfaces homem-máquina de sistemas interativas devem ser o mais flexível possível, permitindo que o usuário a personalize de acordo com seu gosto, pois por mais que se tente e se esforce sempre haverá usuários que ficarão insatisfeitos.

Muitos cientistas têm pesquisado formas de se desenvolver software para interfaces homem-máquina que garantam aceitação por parte dos usuários. Estas formas visam, principalmente, permitir que pessoas que não possuam aprofundado conhecimento sobre computadores e engenharia de software sejam capazes de projetar e programar interfaces. Sistemas para gerenciamento de interfaces ou sistemas para projeto de interfaces de forma interativa têm surgido. No entanto, tais sistemas tem pecado por serem altamente restritivos. Somente interfaces que se enquadram nos casos previstos pelos projetistas de tais sistemas são desenvolvidas com facilidade. Quando se deseja projetar alguma interface que difere daquelas previstas por estes sistemas, o projetista tem de recorrer a um programador experiente para que este, através de métodos *ad-hoc* consiga que o sistema gere a interface desejada, o que vai contra a filosofia destes sistemas. A grande parte dos sistemas de projeto de interfaces utiliza linguagens próprias para especificação de diálogos e do formato da interface. Segundo os fabricantes destes sistemas para projeto de interfaces, a utilização de tais linguagens acelera o processo de desenvolvimento do software. Mas, será que a necessidade de se aprender uma linguagem de uso restrito não irá, na verdade, retardar a produção de um software? A

literatura tem mostrado que os programadores acabam sempre por utilizar bibliotecas ou *toolkits* para desenvolvimento da interface de suas ferramentas.

References

- [Ama92] Lucas M. Amaral. Interface homem-máquina do ambiente de definição de semântica lds. Master's thesis, UFMG, 1992.
- [B⁺87] Bill Bets et al. Goals and objectives for user interface software. *ACM SIG-GRAPH*, 21(2), April 1987. Workshop on Software Tools for User Interface Management.
- [BC91] Len Bass and Joëlle Coutaz. *Developing Software for the User Interface*. SEI Series in Software Engineering. Addison-Wesley Publishing Company, Inc., 1991.
- [BMS86] D.G. Brown, S. Mittal, and M.J. Stefik. Expert systems: Perils and problems. *Communications of the ACM*, September 1986.
- [Fou89] OSF Open Software Foundation. *Motif Style Guide*. Prentice-Hall, 1989.
- [Har89] Rex Hartson. User-interface management control and communication. *IEEE Computer*, January 1989.
- [Lau90] Brenda Laurel. *The Art of Human-Computer Interface Design*. Addison-Wesley Publishing Company, Inc., 1990.
- [Nor83] Donald A. Norman. Design rules based on analyses of human error. *Communications of the ACM*, 26(4), April 1983.
- [SA90] Sun Microsystems, Inc. and AT&T. *OPEN LOOK Graphical User Interface Application Style Guidelines*. Addison-Wesley Publishing Company, Inc., 1990.
- [Sch87] B. Schneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Reading, Mass., 1987.
- [SG86] Robert W. Scheifler and Jim Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2), April 1986.
- [TK89] John C. Thomas and Wendy A. Kellogg. Minimizing ecological gaps in interface design. *IEEE Computer*, January 1989.